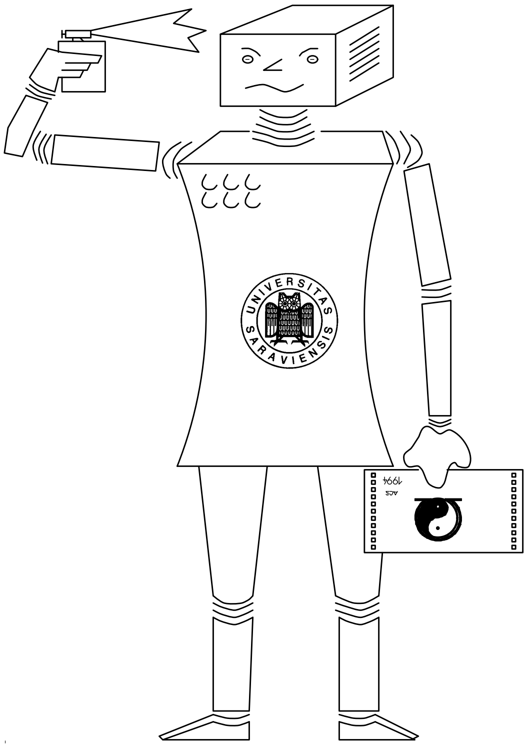
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://js-sfbsun.cs.uni-sb.de/pub/www/



On the Translation of Higher-Order Problems into First-Order Logic

Manfred Kerber

Published as: In Tony Cohn, ed., *Proceedings of the 11th ECAI*, p. 145-149, Amsterdam, The Netherlands, 1994.

ECAI 94. *11th European Conference on Artificial Intelligence* Edited by A. Cohn
Published in 1994 by John Wiley & Sons, Ltd.

On the Translation of Higher-Order Problems into First-Order Logic

Manfred Kerber[1]

Abstract. In most cases higher-order logic is based on the λ-calculus in order to avoid the infinite set of so-called comprehension axioms. However, there is a price to be paid, namely an undecidable unification algorithm. If we do not use the λ-calculus, but translate higher-order expressions into first-order expressions by standard translation techniques, we have to translate the infinite set of comprehension axioms, too. Of course, in general this is not practicable. Therefore such an approach requires some restrictions such as the choice of the necessary axioms by a human user or the restriction to certain problem classes. This paper will show how the infinite class of comprehension axioms can be represented by a finite subclass, so that an automatic translation of finite higher-order problems into finite first-order problems is possible. This translation is sound and complete with respect to a Henkin-style general model semantics.

1 Introduction

First-order logic is a powerful tool for expressing and proving mathematical facts. Nevertheless higher-order expressions are often better suited for the *representation* of mathematical knowledge and in fact almost all mathematical text books rely on some higher-order fragments for expressiveness. This fragment can be obtained directly by using a higher-order logic or indirectly by "implementing" a portion of it in first-order logic with the help of formal set theory. However, each approach has its own advantages and drawbacks and the question of which approach is better is mainly unanswered. For a detailed discussion on mathematical logic and set theory see [18, 21].

Perhaps the most promising approach to operationalizing set theory has been worked out by Robert Boyer et al. [5] and Art Quaife [20] employing the set theory of von Neumann, Bernays, and Gödel, which enables a finite axiomatization in first-order logic. The advantage is the possibility to employ standard first-order theorem provers like Otter [16] for automated deduction and thereby make the reasoning power of first-order theorems available for higher-order theorems too. However, there are some severe drawbacks to using set theory. First, in set theory *every* term denotes a set: there is no syntactic distinction between objects, predicates, or functions, or even more structured entities like natural numbers, planes, or groups. As a consequence, humans as well as automated theorem provers have no syntactic distinction at hand as in higher-order or sorted logics. Second, the key concept of mathematics, namely that of a function, is not a primitive object of set theory, but must be defined in terms of sets, namely as a left-total, right-unique relation, and a relation in turn is defined as a subset of the Cartesian product of two sets, etc. Quaife has shown that it is possible to overcome some of these problems and that it is possible to prove non-trivial theorems in this set theory; however, it might be questioned whether there are more adequate approaches.

Higher-order logic in its full strength, in particular higher-order logic with sorts, is adequate for representing mathematics. Unfortunately one has to pay a price, namely that the notions of *truth* and *provability* no longer coincide [9]. In order to be able to operationalize higher-order logic Leon Henkin weakened the standard semantics of higher-order logic [11] and thereby showed the way to complete calculi for higher-order logic, in particular for Alonzo Church's simple theory of types [6]. Nevertheless, it is hard to build a corresponding theorem prover. For instance, an adequate higher-order theorem prover has to incorporate higher-order unification as introduced by Gérard Huet, which has unpleasant properties: For the general case, unification is undecidable and there is no complete set of most general unifiers. Another problem of higher-order resolution theorem proving is the necessity to apply so-called splitting rules (compare [14]). Furthermore the treatment of the extensionality of functions is not satisfactorily solved. Perhaps the most advanced system for higher-order logic is the TPS system of Peter Andrews and his group [3].[2]

A third approach[3] is based on a translation of higher-order expressions into (many-sorted) first-order logic. For second-order logic this approach was introduced by Herbert B. Enderton [7]. For general higher-order logic, Lawrence J. Henschen [12] extended the first-order operationalization by constructs for handling the so-called comprehension axioms. This translation is insofar incomplete as it does not handle extensionality. In [15], the author gives a translation into standard many-sorted first-order logic that is proved to be complete with respect to a Henkin-style general model semantics. This translation method cannot be used fully automatically, since for its completeness it requires a priori infinitely many comprehension axioms. That is, the necessary ones must be selected manually. In this paper we show how the comprehension axioms can be finitely represented for a finitely typed signature. That is, (restricted) higher-order logic can be mapped into (many-sorted) first-order logic. The (only) price that has to be paid is a finite set of additional first-order formulae.

In the following we introduce the syntax and semantics of

[1] Fachbereich Informatik, Universität des Saarlandes, 66041 Saarbrücken, Germany, e-mail: kerber@cs.uni-sb.de. This work was supported by the Deutsche Forschungsgemeinschaft, SFB 314 (D2).

[2] For the direct operationalization of higher-order logic there is an alternative to Church's λ-calculus, namely the calculus of combinators, see e.g. [13].

[3] In another approach, first-order formulations of a class of second-order problems can be achieved by eliminating quantified second-order predicates with the help of the resolution calculus. Compare e.g. [17, 8].

ECAI 94. *11th European Conference on Artificial Intelligence* Edited by A. Cohn
Published in 1994 by John Wiley & Sons, Ltd.

our higher-order logic, then we present the finite representation of the comprehension axioms, shortly recall the translation into first-order logic, and present an Otter proof for Cantor's theorem, a standard example for a higher-order theorem.

2 Higher-Order Logic

The syntax of our higher-order logic is essentially based on Alonzo Church's simple theory of types [6]. For details on higher-order logic the reader is referred to the excellent presentation in [2]. For the finite representability of the comprehension axioms the finiteness of the order of the logic is crucial, that is, we can translate problems, formulated in an n-th order logic $\mathcal{L}^n$, but not those requiring a higher-order logic of unlimited order. Since we have to employ a variant of the standard formalism we will recall our concrete syntax in the next section.

The Syntax

As usual the types of the higher-order logic are inductively constructed from the base types (of order 0) ι, the type of the individuals, and o, the type of the truth values, by the construction: if $\tau_1, \ldots, \tau_m$, and τ are types (with $m \geq 1$ and τ_i being unequal to o), then $(\tau_1 \times \cdots \times \tau_m \to \tau)$ is a type of order 1 + maximum of the orders of $\tau_1, \ldots, \tau_m, \tau$. It denotes the type of m-ary functions with arguments of type $\tau_1, \ldots, \tau_m$, respectively, and value of type τ.

By this definition we exclude – unlike Church and Andrews [6, 2] – types such as $(o \to o)$. These types give rise to special problems in the translation into first-order logic, because they are essentially on the level of connectives. Therefore and since we have no λ-binder in our restricted language it is not possible to define the connectives $\neg$ and $\wedge$ and the quantifier $\forall$, and hence they must be introduced as primitives. Nevertheless we conjecture that the languages $\mathcal{L}^n$ – defined below – are adequate for expressing mathematical facts. For instance we can have predicates like *ordered_group*$(G, +, \leq)$ of type $((\iota \to o) \times (\iota \times \iota \to \iota) \times (\iota \times \iota \to o) \to o)$. In fact in all our examples from mathematical textbooks, this was no serious restriction.

The signature $\mathcal{S}$ of the logics $\mathcal{L}^{2n}$ or $\mathcal{L}^{2n-1}$ consists first of a *finite* set of types, called $\mathfrak{T}$ such that each type in $\mathfrak{T}$ has an order of n or less and second for each type in $\mathfrak{T}$ of a possibly empty set of constant symbols. That is, $\mathcal{S} = \langle \mathfrak{T}, \{\mathcal{S}_\tau\}_{\tau \in \mathfrak{T}} \rangle$. $\mathcal{L}^{2n-1}$ is the subset of $\mathcal{L}^{2n}$ such that no variable of order n is quantified on.

We assume $\mathfrak{T}$ to be subtype-closed, that is, for instance, if $(\iota \to \iota) \in \mathfrak{T}$ then $\iota \in \mathfrak{T}$. Furthermore we assume that there are infinitely many variable symbols for each type in $\mathfrak{T}$.

The terms of a logic $\mathcal{L}^n$ are defined as usual

1. Every variable or constant of a type τ is a *term* of type τ.
2. If $f_{(\tau_1 \times \cdots \times \tau_m \to \tau)}, t_{\tau_1}, \ldots, t_{\tau_m}$ are terms of the type indicated by their subscripts, then $f_{(\tau_1 \times \cdots \times \tau_m \to \tau)}(t_{\tau_1}, \ldots, t_{\tau_m})$ is a *term* of type τ.

Terms of type o are atomic formulae.[4] Formulae in general are either atomic or built up by composing them with the means of connectives and quantifiers, that is, if φ and ψ are formulae and x is a variable of any type in $\mathfrak{T}$, then $(\neg\varphi)$, $(\varphi \wedge \psi)$, and $(\forall x \varphi)$ are formulae. As long as there is no danger of confusion we often omit parentheses.

1 Remark: As usual one can define (on a meta-level) $\vee$, $\Rightarrow$, $\Leftrightarrow$, and $\exists$ in terms of $\neg$, $\wedge$, and $\forall$ and use formulae containing these symbols as abbreviations. We have excluded all λ-expressions in our logic. If they are included, translations into first-order logic end up in higher-order theorem proving with the above mentioned problems of higher-order logic such as undecidable unification. They are not necessary for formulating mathematics, since we can introduce a name for each λ-expression. The very idea of λ-expressions is not to increase the expressiveness of higher-order logic, but to eliminate the so-called comprehension axioms in the operationalization. ∎

The Semantics

The usual way to give a Tarski-semantics is mapping each term of type τ into a non-empty universe $\mathcal{D}_\tau$ (respecting the usual homomorphic conditions) such that the universe of a functional type is just the set of the corresponding functions,[5] that is, $\mathcal{D}_{(\alpha_1 \times \cdots \times \alpha_n \to \beta)} = \mathcal{F}(\mathcal{D}_{\alpha_1} \times \cdots \times \mathcal{D}_{\alpha_n}, \mathcal{D}_\beta)$. However, as proved in Kurt Gödel's famous incompleteness theorem [9] with respect to such a *strong* (generally intended) semantics no complete calculus can be found in which it can be calculated whether a formula φ follows from a formula set Γ, written as $\Gamma \models\!\!\!\models \varphi$. In order to extend the ideas of compactness, completeness and soundness from first-order logic to higher-order logic, Leon Henkin has generalized the semantics to a weak version that allows for complete calculi and which is generally taken – in some variant (e.g. [1]) – as the base of higher-order theorem proving. Henkin's very idea is to replace the "=" in the relation of the universes above by a "$\subseteq$", that is the restriction is only $\mathcal{D}_{(\alpha_1 \times \cdots \times \alpha_n \to \beta)} \subseteq \mathcal{F}(\mathcal{D}_{\alpha_1} \times \cdots \times \mathcal{D}_{\alpha_n}, \mathcal{D}_\beta)$. This leads to a weakened model relation and to a weakened consequence relation $\models$, for which holds: if $\Gamma \models \varphi$ then $\Gamma \models\!\!\!\models \varphi$. If you make only this one change you get essentially a first-order semantics for higher-order logic. However, in this form the semantics is too weak, since it is possible to define functions that have no counterpart in the semantics. In order to overcome this, a set of comprehension axioms Υ is assumed so that if $\Gamma \cup \Upsilon_{\mathfrak{T}} \models \varphi$ then $\Gamma \models\!\!\!\models \varphi$.

2 Definition (Comprehension Axioms): The *comprehension axioms* $\Upsilon_{\mathfrak{T}}$ are the following formulae (a slightly different set can be found in [2, p.156]):
For every term (or formula) t of type τ of which the free variables are at most the different variables $x_1, \ldots, x_m, y_1, \ldots, y_k$ of type $\tau_1, \ldots, \tau_m, \sigma_1, \ldots, \sigma_k$ (all types τ, τ_i, σ_j being in $\mathfrak{T}$), in particular f does not occur free in t:
$\forall y_1 \ldots \forall y_k \exists f_{(\tau_1 \times \cdots \times \tau_m \to \tau)} \forall x_1 \ldots \forall x_m\ f(x_1, \ldots, x_m) \doteq t$. ∎

These axioms allow for instance that higher-order variables such as P in the induction formula $\forall P_{(\iota \to o)} P(0) \wedge (\forall n_\iota P(n) \Rightarrow P(s(n))) \Rightarrow (\forall n_\iota P(n))$, the atomic expression $P(n)$ can be instantiated by a formula like $\Sigma_{i=1}^n i = n \cdot (n+1)/2 \wedge \mathrm{odd}(s(n))$.

[4] We assume fixed binary predicates $\doteq$ of type $(\tau \times \tau \to o)$ for any type τ with the usual fixed semantics. $\doteq$ of type $(o \times o \to o)$ is taken synonymous for the connective $\Leftrightarrow$.

[5] By $\mathcal{F}(A, B)$ we denote the set of all functions from A to B.

3 Generator Axioms

Unfortunately the set of comprehension axioms is a priori infinite, even for a finite signature (that is even for finitely many types in $\mathfrak{T}$). Therefore these axioms cannot be given to an automated theorem prover in total. The idea is to replace the set Υ by a finite set GENAX such that $\Gamma \cup \Upsilon \models \varphi$ iff $\Gamma \cup \text{GENAX} \models \varphi$.

In this section we construct a finite set of axioms GENAX that can replace the infinite set of comprehension axioms Υ. The construction of this set and the argument that these axioms can replace Υ is analogous to that of John von Neumann in the construction of his set theory, where he was able to eliminate the infinite set of comprehension axioms. Since the predicate and function symbols in set theory (like $\in$ and $\cap$) are all known in advance, a fixed set of 18 axioms is sufficient to formalize the whole set theory (Compare [10]. The axioms B1 through B8 have just the structure of comprehension axioms. Nevertheless the possibility to construct the whole theory of sets on this finite axiomatization is far from being trivial, in Zermelo-Fraenkel set theory, for instance, an infinite set of axioms is necessary.) However, in our case the set of types is not known in advance, therefore we can only calculate the axioms as a function of the set of types $\mathfrak{T}$. Note that $\mathfrak{T}$ depends not directly on the occurring function and predicate symbols but only on the occurring types.

Let $\mathcal{S} = \langle \mathfrak{T}, \{\mathcal{S}_\tau\}_{\tau\in\mathfrak{T}} \rangle$ be the signature of $\mathcal{L}^n$. Then it is sufficient to consider as comprehension axioms the following set GENAX of axioms.

3 CONVENTION: Since we have to deal a lot with tuples, we shall use the following notation for the rest of this paper: $\overline{\tau_n} := \tau_1 \times \cdots \times \tau_n$, $\forall \overline{x_n} := \forall x_1 \ldots \forall x_n$, and so on. ■

For each type in $\mathfrak{T}$ we have four groups of axioms (we assume all terms to be well-typed, but we omit the indexes where possible in order to avoid a mess of symbols).[6]

PERM In this class are axioms that allow one to permute the arguments of parameters (that are function or predicate symbols)[7] as in $g(x,y) = f(y,x)$.

Let $\overline{\tau_n}$ be given and let π be a permutation on $\{1,\ldots,n\}$, such that π permutes two neighbouring elements, that is, $\pi(i) = i+1$ and $\pi(i+1) = i$ for $i < n$ then the following formula is in PERM:

$$\forall f_{\overline{\tau_n}\to\tau'} \exists g_{\tau_{\pi(1)}\times\cdots\times\tau_{\pi(n)}\to\tau} \forall x_{\tau_1} \ldots \forall x_{\tau_n} \; g(x_{\tau_{\pi(1)}},\ldots,x_{\tau_{\pi(n)}}) = f(x_{\tau_1},\ldots,x_{\tau_n})$$

JOIN In this class two arguments can be identified as in $g(x) = f(x,x)$.

Let $\tau_1,\ldots,\tau_n$ be given with $\tau_1 = \tau_2$ then the following formula is in JOIN:

$$\forall f_{\tau_1\times\cdots\times\tau_n\to\tau} \exists g_{\tau_1\times\tau_3\times\cdots\times\tau_n\to\tau} \forall x_{\tau_1} \forall x_{\tau_3} \ldots \forall x_{\tau_n} \; g(x_{\tau_1},x_{\tau_3},\ldots,x_{\tau_n}) = f(x_{\tau_1},x_{\tau_1},x_{\tau_3},\ldots,x_{\tau_n})$$

PROJ By these axioms the projection of a function with respect to one fixed argument can be calculated, as for instance in $g(y) = f(x,y)$.

Let $\tau_1,\ldots,\tau_n$ be given, then the following axiom is in PROJ:

$$\forall x_{\tau_1} \forall f_{\tau_1\times\cdots\times\tau_n\to\tau} \exists g_{\tau_2\times\tau_3\times\cdots\times\tau_n\to\tau} \forall x_{\tau_2} \ldots \forall x_{\tau_n} \; g(x_{\tau_2},\ldots,x_{\tau_n}) = f(x_{\tau_1},\ldots,x_{\tau_n})$$

NEST By these axioms it is possible to nest parameters. Nesting is a little more complicated than the axioms above. We would like to add the following axioms: Let τ, τ' be given, then there is an axiom:

$$\forall f_{\tau_1\times\cdots\times\tau_n\to\tau} \forall h_{\sigma_1\times\cdots\times\sigma_m\to\tau_1} \exists g_{\sigma_1\times\cdots\times\sigma_m\times\tau_2\times\cdots\times\tau_n\to\tau} \forall y_{\sigma_1} \ldots \forall y_{\sigma_m} \forall x_{\tau_2} \ldots \forall x_{\tau_n} \; g(y^1,\ldots,y^m,x^2,\ldots,x^n) = f(h(y^1,\ldots,y^m),x^2,\ldots,x^n)$$

However, this axiom is not necessarily in $\Upsilon_{\mathfrak{T}}$ since the type of g is not necessarily in $\mathfrak{T}$. Therefore, we define NEST as the set of all possible formulae in $\Upsilon_{\mathfrak{T}}$ that can be generated from the above axioms by projecting variables or by multiple occurrences of variables as done by the rules JOIN and PROJ (In the case of $\tau = o$ variables can be eliminated by axioms from QUANT too). Concretely that means, NEST is the set:

$$\{\forall y^1,\ldots,y^j,x^2,\ldots,x^l,f,h \; \exists g_\tau \; \forall y^{j+1},\ldots,y^s,x^{l+1},\ldots,x^r \; g(y^{j+1},\ldots,y^s,x^{l+1},\ldots,x^r) = f(h(y^1,y^2,\ldots,y^j,y^{j+1},\ldots,y^{j+1},\ldots,y^s,\ldots,y^s), x^2,x^3,\ldots,x^l,x^{l+1},\ldots,x^{l+1},\ldots,x^r,\ldots,x^r) | \tau \in \mathfrak{T}\}$$

Furthermore we need the following axioms for conjunction, negation, and quantification:

CONJ Let p and p' be two n- and m-ary predicate symbols, then we have an axiom:

$$\forall p_{\overline{\tau_n}\to o} \forall p'_{\overline{\tau'_m}\to o} \exists q_{\overline{\tau_n}\times\overline{\tau'_m}\to o} \forall \overline{x_n} \forall \overline{y_m} \; q(\overline{x_n},\overline{y_m}) \Leftrightarrow p(\overline{x_n}) \wedge p'(\overline{y_m}).$$

As in the case of the NEST axioms we might fall out of $\Upsilon_{\mathfrak{T}}$ by this axiom, then we take only the corresponding axioms that can be constructed by the projection of variables or by multiple occurrences on the right hand side.

NEG $\forall p_{\overline{\tau_n}\to o} \exists q_{\overline{\tau_n}\to o} \forall \overline{x_n} \; q(\overline{x_n}) \Leftrightarrow \neg p(\overline{x_n})$

QUANT Let $\overline{\tau_n}, \sigma$ be types then the following formula is in QUANT:

$$\forall p \exists q \forall \overline{x_n} \; q(\overline{x_n}) \Leftrightarrow (\forall y \; p(\overline{x_n}, y))$$

We define GENAX := PERM $\cup$ JOIN $\cup$ PROJ $\cup$ NEST $\cup$ CONJ $\cup$ NEG $\cup$ QUANT.

By the syntactic structure of the axioms, it is easy to see that each formula in GENAX is also in Υ. Hence we have the following two lemmas.

4 LEMMA: *For each finitely typed signature $\mathcal{S}$, the set of generator axioms* GENAX *is finite.* ■

5 LEMMA: *Each generator axiom in* GENAX *is in* Υ. ■

6 THEOREM: *For each axiom v in Υ,* GENAX $\models v$

Proof: If v is already in GENAX we are done. Else, we show the assumption by induction on the structure of v. Since v is in Υ it has the structure:

$$\forall \overline{y_k} \exists f_{(\overline{\tau_m}\to\tau)} \forall \overline{x_m} \; f(\overline{x_m}) = t$$

We make a case analysis on the structure of t, and show inductively that v follows from the generator axioms.

- Let t be a composed formula (in particular we have $\tau = o$). Let t be a negation, that is, $t = \neg\psi$. Assume by induction

[6] In our description, we keep the number of axioms minimal. In practical applications it might be preferable to add further axioms. For instance in the case of the permutations, we take only such permutations that change the position of two neighbours, by composition we get all non-trivial permutations. Therefore we do with $n-1$ axioms where elsewhere $n!-1$ would be necessary.

[7] The axioms PERM correspond to B6, B7, and B8 in [10], for instance.

hypotheses that the formula $\forall\overline{y_k}\exists f'_{(\overline{\tau_m}\to\tau)}\forall\overline{x_m}\ f'(\overline{x_m})\dot{=}\psi$ follows from GENAX. It is easy to see that then $\forall\overline{y_k}\exists f_{(\overline{\tau_m}\to\tau)}\forall\overline{x_m}\ f(\overline{x_m})\dot{=}\neg\psi$ follows from GENAX$\cup\left\{\forall\overline{y_k}\exists f'_{(\overline{\tau_m}\to\tau)}\forall\overline{x_m}\ f'(\overline{x_m})\dot{=}\psi\right\}$ by instantiating p to f' in NEG (i.e. $\forall\overline{y}\exists f\forall\overline{x} f(\overline{x})\dot{=}\neg f'(\overline{x})$). Hence v follows from GENAX.

In a similar manner we can conclude by structural induction that for other composed formulae t, that is, conjunctions or quantifications, by the application of axioms from CONJ or QUANT, v follows from GENAX.

- Let t be a non composed term. By the axioms in PERM, JOIN, and PROJ we can construct all combinations of free variables occurring in a term t of term depth one. The NEST axioms allow for successive nesting of terms. ■

4 A Standard Translation from Higher-Order to First-Order Logic

In this section we briefly recall the general translation method from higher-order to many-sorted first-order logic by employing a so-called "apply" function that reifies function and predicate symbols (compare [15]).

7 DEFINITION (STANDARD TRANSLATION Θ):
Let $\mathcal{S}$ be a higher-order signature, the translation of the corresponding higher-order logic into many-sorted logic is given as follows: every type τ of higher-order logic is associated to a sort $\tilde{\tau}$. The sort relation is trivial insofar as all sorts are disjoint. Each constant or variable of a certain type is mapped by a mapping Θ onto a corresponding constant or variable of the associated sort (In order to avoid a lot of redundant names, we use the same names for source and target). Furthermore we have for each non-base type $\sigma = (\tau_1 \times\cdots\times \tau_m \to \tau)$ in $\mathfrak{T}$, a new free function symbol $\alpha_{\tilde{\sigma}}$ of sort $(\tilde{\sigma}\times\tilde{\tau_1}\times\cdots\times\tilde{\tau_m}\to\tilde{\tau})$.[8]
For terms we define a mapping Θ inductively by:

T1 For a term with an m-ary function term f of type τ as top expression we define
$\Theta(f(t_1,\ldots,t_m)) = \alpha_{\tilde{\tau}}(\Theta(f),\Theta(t_1),\ldots,\Theta(t_m))$

For non-atomic formulae we define Θ inductively by:

F1 For a conjunction we define
$\Theta(\varphi_1\wedge\varphi_2) = \Theta(\varphi_1)\wedge\Theta(\varphi_2)$

F2 For a negation we define
$\Theta(\neg\varphi) = \neg\Theta(\varphi)$

F3 For a quantified formula we define
$\Theta(\forall x\varphi) = \forall\Theta(x)\Theta(\varphi)$

$\Xi_{\mathfrak{T}}$ is the set consisting of the following formulae:

$\Xi_{\mathfrak{T}}$ For every non basic type σ in $\mathfrak{T}$ with $\sigma = (\tau_1\times\cdots\times\tau_m\to\tau)$:
$\forall f_{\tilde{\sigma}}\forall g_{\tilde{\sigma}}(\forall x^1_{\tilde{\tau}_1},\ldots,\forall x^m_{\tilde{\tau}_m}$
$\alpha_{\tilde{\sigma}}(f,x^1,\ldots,x^m)\dot{=}\alpha_{\tilde{\sigma}}(g,x^1,\ldots,x^m)) \Rightarrow f\dot{=}g$ ■

From [15] follows in combination with lemma 5 and theorem 6 the main result, which allows to translate a finite higher-order problem into a finite first-order problem.

8 THEOREM: *In order to show that a formula φ follows from a set of formulae Γ in a higher-order logic with types $\mathfrak{T}$ with respect to (the above defined Henkin-style) general model semantics, it is necessary and sufficient that in many-sorted first-order logics holds*

$$\Theta(\Gamma)\cup\Theta(\text{GENAX}_{\mathfrak{T}})\cup\Xi_{\mathfrak{T}}\models\Theta(\varphi)$$ ■

5 Example

A standard example for showing the behaviour of a theorem proving system on higher-order theorems is Cantor's theorem that there is no surjective mapping from a set A onto its powerset $\wp(A)$. Higher-order theorem provers like TPS are good at proving this theorem. In a previous work we have shown how this theorem can be proved using a comprehension axiom [15]. It has also been translated to set theory in [20], and proved in von Neumann-Bernays-Gödel set theory [19, 10, 4] as proposed in [5]. However, in the proof in [15] as well as in that of [20] the key idea, namely the diagonalization construct, has been given by the human user.[9]

In the following we give an Otter proof of the theorem without telling it the key idea. The diagonalization construct can easily be built up by applying the constructor axioms for negation and conjunction. Otter is not directly well-suited for the formulation of such a higher-order problem, since it does not support any sorts. However, it is not to hard to see that in this case, the sort information can be omitted, since the sort of each expression can be fixed by the position of the expression in an apply function or predicate symbol. This is possible because of the absence of equality relations. Therefore we have chosen a formulation in which the extensionality is already expanded.[10] The theorem has the following form:
$\forall s_{\iota\to o}\neg\exists g_{\iota\to(\iota\to o)}\forall f_{\iota\to o} f\subseteq s\Rightarrow(\exists j_\iota s(j)\wedge g(j)=f)$
The $\alpha_{\iota\times\iota\to o}$ predicate is called in Otter syntax `aixito`, analogously the other α functions.

```
set(hyper_res).
...
formula_list(axioms).
   %PERM
   (all f (exists g (all x (all y (aixito(g,x,y) <-> aixito(f,y,x)))))).
   %two JOIN axioms
   (all f (exists g (all x (aito(g,x) <-> aixito(f,x,x))))).
   (all f (exists g (ao(g) <-> (all x aito(f,x))))).
   %PROJ axiom
   (all x (all f (exists g (all y (aito(g,y) <-> aixito(f,x,y)))))).
   (all x (all f (exists g (all y (ao(g) <-> aito(f,x)))))).
   %no NEST axioms
   %CONJ
  (all x (all y (all p (all pp (exists q (all u (all v (aixito(q,u,v) <->
                              (aixito(p,u,v) & aixito(pp,x,y)))))))))).
  (all x (all y (all p (all pp (exists q (all u (all v (aixito(q,u,v) <->
                              (aixito(p,x,u) & aixito(pp,v,y)))))))))).
  (all x (all p (all pp (exists q (all u (all v (aixito(q,u,v) <->
                              (aixito(p,u,v) & aixito(pp,x,x))))))))).
  (all x (all p (all pp (exists q (all u (all v (aixito(q,u,v) <->
                              (aixito(p,u,x) & aixito(pp,v,x))))))))).
  (all p (all pp (exists q (all u (all v (aixito(q,u,v) <->
                              (aixito(p,u,u) & aixito(pp,v,v)))))))).
  (all p (all pp (exists q (all u (all v (aixito(q,u,v) <->
                              (aixito(p,u,v) & aixito(pp,u,v)))))))).
```

[8] The function ˜ must map the types onto new names.

[9] In Quaife's approach this hint has been given for efficiency reasons.

[10] The translation of unsorted higher-order logic into many-sorted first-order logic results in a flat sort structure, so a sorted first-order theorem prover is more adequate. However, since the sorts are flat, an encoding in standard first-order logic is possible without the necessity of relativizing the formulae. We have chosen Otter and not a prover like MKRP for this example, since we want to relate our work to Quaife's.

```
  (all x (all p (all pp (exists q (all y (aito(q,y) <->
                                  (aito(p,x) & aito(pp,y))))))))).
  (all x (all p (all pp (exists q (all y (aito(q,y) <->
                                  (aito(p,y) & aito(pp,x))))))))).
  (all p (all pp (exists q (all y (aito(q,y) <->
                              (aito(p,y) & aito(pp,y)))))))).
  (all p (all pp (exists q (ao(q) <-> (ao(p) & ao(pp)))))).
  %NEG
 (all p (exists q (all x (all y (aixito(q,x,y) <-> - aixito(p,x,y)))))).
  (all p (exists q (all x (aito(q,x) <-> - aito(p,x))))).
  (all p (exists q (ao(q) <-> - ao(p)))).
  %QUANT
  (all p (exists q (all x (aito(q,x) <-> (all y aixito(p,x,y)))))).
  %no EXTENSIONALITY AXIOMS since equality eliminated
  %in problem
end_of_list.
formula_list(sos).
  %THEOREM
  (all s (- exists g (all f ((all x (aito(s,x) -> aito(f,x))) ->
    (exists j (aito(s,j) & (all x (aixito(g,j,x) <-> aito(f,x))))))))).
end_of_list.
----> EMPTY CLAUSE at   1.04 sec ----> 140 [hyper,135,43,135] .
---------------- PROOF ----------------
33 [] - aito(f14(x,x20,x21),y) — aito(x21,x).
43 [] - aito(f18(x27),x) — - aito(x27,x).
49 [] aito(x30,f23(x30,x31)) — aito(x30,f24(x30,x31)).
52 [] - aito(x31,f23(x30,x31)) — aito(x30,f24(x30,x31)).
113 [hyper,52,49] aito(x,f24(x,x)).
135 [hyper,113,33] aito(x,y).
140 [hyper,135,43,135] .
```

In our formulation the proof of Cantor's theorem is much easier to find for Otter than in Quaife's approach (140 resolvents versus 700, 19 axioms versus 352 possibly applicable theorems, 1 second versus 10 seconds of run time) although we have not given the hint how to construct the diagonal. Therefore there is a legitimate hope that the translation method is advantageous for a whole class of problems, in particular since refinements such as the employment of sorts are possible. (A more detailed comparison is in work and will be published as a SEKI-Report.)

6 Conclusion

We have shown how the comprehension axioms can be finitely represented. This makes an effective use of first-order theorem provers for problems formulated in a higher-order logic possible and sheds some light on the difference between first-order and higher-order logic with its first-order semantics as introduced by Leon Henkin. Standard first-order theorem provers can be used by reifying predicate and function symbols by so-called "apply"-constructs (write $\alpha(f,x)$ instead of $f(x)$). In order to be as complete as possible you have to add extensionality axioms (there are only finitely many for finitely typed signatures) and comprehension axioms (there are a priori infinitely many even for finitely typed signature). However, as seen the infinitely many comprehension axioms can be finitely represented by the generator axioms GENAX. Thereby it is possible to employ standard many-sorted first-order theorem provers. For cases without equations a first-order theorem prover for unsorted logic does as well.

In order to optimize the approach, it would be useful to extend it to higher-order sorted logics (this should be straightforward). The main drawback of the whole attempt, namely the necessity to add a lot of non-trivial axioms, may be compensated by configuring the first-order theorem prover in such a way that the generator axioms are only used in a very reserved manner, for instance, by assigning high weights to them. As practice has shown, for many theorems no comprehension axioms are necessary at all, so it is a good idea to try first whether the theorem can be proved without using any and only if a proof is not found in a reasonable amount of time, the generator axioms are added.

I like to thank Dan Nesmith and the anonymous referees for many useful comments.

REFERENCES

[1] Peter B. Andrews, 'General models and extensionality', *Journal of Symbolic Logic*, **37**(2), 395–397, (1972).

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

[3] Peter B. Andrews, Sunil Issar, Dan Nesmith, and Frank Pfenning, 'The TPS theorem proving system', in *Proc. of the 10th CADE*, ed., M. E. Stickel, pp. 641–642, Kaiserslautern, Germany, (1990). Springer Verlag. LNAI 449.

[4] Paul Bernays, 'A system of axiomatic set-theory', *Journal of Symbolic Logic*, **6**, 1–17, (1941).

[5] Robert Boyer, Ewing Lusk, William McCune, Ross Overbeek, Mark Stickel, and Lawrence Wos, 'Set theory in first-order logic: Clauses for Gödel's axioms', *Journal of Automated Reasoning*, **2**, 287–327, (1986).

[6] Alonzo Church, 'A formulation of the simple theory of types', *Journal of Symbolic Logic*, **5**, 56–68, (1940).

[7] Herbert B. Enderton, *A Mathematical Introduction to Logic*, Academic Press, 1972.

[8] Dov Gabbay and Hans Jürgen Ohlbach, 'Quantifier eliminiation in second-order predicate logic', in *Proc. of KR'92*, eds., B. Nebel, C. Rich, and W. Swartout, pp. 425–435, Cambridge, Massachusetts, USA, (1992). Morgan Kaufmann.

[9] Kurt Gödel, 'Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I', *Monatshefte für Mathematik und Physik*, **38**, 173–198, (1931).

[10] Kurt Gödel, *The Consistency of the Axiom of Choice and of the Generalized Continuum-Hypothesis with the Axioms of Set Theory*, Princeton University Press, 1940.

[11] Leon Henkin, 'Completeness in the theory of types', *Journal of Symbolic Logic*, **15**, 81–91, (1950).

[12] Lawrence J. Henschen, 'N-sorted logic for automatic theorem-proving in higher-order logic', in *Proc. of the Annual Conference of the ACM*, pp. 71–81, Boston, Massachusetts, USA, (1972). Association for Computing Machinery, Washington, DC, USA, ACM Press.

[13] J. R. Hindley and J. P. Seldin, *Introduction to Combinators and λ-Calculus*, Cambridge University Press, 1986.

[14] Gérard Huet, *Constraint Resolution: A Complete Method for Higher-Order Logic*, Ph.D. dissertation, Case Western Reserve University, 1972.

[15] Manfred Kerber, 'How to prove higher order theorems in first order logic', in *Proc. of the 12th IJCAI*, eds., J. Mylopoulos and R. Reiter, pp. 137–142, Sydney, (1991). Morgan Kaufman.

[16] William McCune, 'Otter 2.0', in *Proc. of the 10th CADE*, ed., M. E. Stickel, pp. 663–664, Kaiserslautern, Germany, (1990). Springer Verlag. LNAI 449.

[17] John Threecivelous Minor, *Proving a Subset of Second-Order Logic with First-Order Proof Procedures*, Ph.D. dissertation, University of Texas, Austin, Texas, USA, 1979.

[18] Gregory H. Moore, 'Beyond first-order logic: The historical interplay between mathematical logic and axiomatic set theory', *History and Philosophy of Logic*, **1**, 95–137, (1980).

[19] John von Neumann, 'Die Axiomatisierung der Mengenlehre', *Mathematische Zeitschrift*, **27**, 669–752, (1928).

[20] Art Quaife, 'Automated deduction in von Neumann-Bernays-Gödel set theory', *Journal of Automated Reasoning*, **8**(1), 91–146, (1992).

[21] Stewart Shapiro, 'Second-order languages and mathematical practice', *Journal of Symbolic Logic*, **50**(3), 714–742, (1985).