A Structured Set of Higher-Order Problems

Christoph E. Benzmüller and Chad E. Brown

Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany
`www.ags.uni-sb.de/{~chris, ~cebrown}`

Abstract. We present a set of problems that may support the development of calculi and theorem provers for classical higher-order logic. We propose to employ these test problems as quick and easy criteria preceding the formal soundness and completeness analysis of proof systems under development. Our set of problems is structured according to different technical issues and along different notions of semantics (including Henkin semantics) for higher-order logic. Many examples are either theorems or non-theorems depending on the choice of semantics. The examples can thus indicate the deductive strength of a proof system.

1 Motivation: Test Problems for Higher-Order Reasoning Systems

Test problems are important for the practical implementation of theorem provers as well as for the preceding theoretical development of calculi, strategies and heuristics. If the test theorems can be proven (resp. the non-theorems cannot) then they ideally provide a strong indication for completeness (resp. soundness). Examples for early publications providing first-order test problems are [21,29,23]. For more than decade now the TPTP library [28] has been developed as a systematically structured electronic repository of first-order test problems. This repository together with the yearly CASC theorem prover competitions [24] significantly supported the improvement of first-order and propositional reasoning systems. Unfortunately, a respective library of higher-order test problems is not yet available.

This paper presents a small set of significant test problems for classical higher-order logic that may guide the development of higher-order proof systems. These test problems are relevant for both automated and interactive higher-order theorem proving. Even some of our simpler theorems may be difficult to prove interactively. Examples are our problems 15(a): $p_{o\to o}\,(a_o \wedge b_o) \Rightarrow p\,(b \wedge a)$ and 16: $(p_{o\to o}\,a_o) \wedge (p\,b_o) \Rightarrow (p\,(a \wedge b))$.

Most of the examples presented here are chosen to be a simple representative of some particular technical or semantical point. We also include examples illustrating real challenges for higher-order theorem provers. Our work is relevant in the first place for theorem proving in classical higher-order logic. However, many of our examples also carry over to other logics such as intuitionistic higher-order logic. Most of the presented test problems evolved from experience gained in the development of the higher-order theorem provers TPS [5] and LEO [10,7]. Some of the examples and (many others) have been also discussed in other publications on classical higher-order logic, e.g. [15,17,6,1,4]. The novel contribution of this paper is not the test problems per se, but the connection of these examples with the particular model classes in which they are valid (resp. invalid) and their assemblage into a comprehensive set.

We structure many of our examples along two dimensions. The examples are theorems or non-theorems depending on these dimensions.

Extensionality provides one dimension in which we can vary semantics. Assuming Henkin semantics, for instance, most of our examples denote theorems. If we choose a weaker semantics, for instance, by omitting Boolean extensionality, then some test problems become non-theorems providing a test case for soundness with respect to this more general notion of semantics (in which fewer propositions are valid). By varying extensionality, we have defined a landscape of eight higher-order model classes and developed abstract consistency methods and model existence results in [8,9]. This landscape of higher-order model classes and the corresponding abstract consistency framework provides much needed support for the theoretical analysis of the deductive power of calculi for higher-order logic. The test problems we introduce in this paper provide quick and easy test criteria for the soundness and completeness of proof systems with respect to these model classes. Testing a proof system with our examples should thus precede a formal, theoretical soundness and completeness analysis with the abstract consistency methodology introduced in [8,9].

Set comprehension provides another dimension along which one can vary semantics. In [14] different model classes are defined depending on the logical constants which occur in the signature. Since many sets are only definable in the presence of certain logical constants, this provides a way of varying the sets which exist in a model. In this paper, we provide examples of theorems which are only provable if one can use certain logical constants for instantiations. In implementations of the automated theorem provers TPS and LEO the problem of instantiating set variables corresponds to the use of *primitive substitutions* described in [14,2,3].

Section 2 introduces the syntax of classical higher-order logic following Church [15]. Section 3 presents some first test problems for pre-unification and quantifier dependencies. In Section 4 we review a landscape of higher-order semantics that distinguishes higher-order models with respect to various combinations of Boolean extensionality, three forms of functional extensionality and different signatures of logical constants. Section 5 provides test problems that are structured according to the introduced landscape of model classes. Section 6 presents some more complex test problems.

2 Classical Higher-Order Logic

As in [15], we formulate higher-order logic ($\mathcal{HOL}$) based on the simply typed λ-calculus. The set of simple types $\mathcal{T}$ is freely generated from basic types o and ι using the function type constructor $\rightarrow$.

For formulae we start with a set $\mathcal{V}$ of (typed) variables (denoted by $X_\alpha, Y, Z, \ldots$) and a signature Σ of (typed) constants (denoted by $c_\alpha, f_{\alpha\rightarrow\beta}, \ldots$). We let $\mathcal{V}_\alpha$ (Σ_α) denote the set of variables (constants) of type α. A signature Σ of constants may include logical constants from the set $\overline{\Sigma}$ defined by

$$\{\top_o, \bot_o, \neg_{o\rightarrow o}, \wedge_{o\rightarrow o\rightarrow o}, \vee_{o\rightarrow o\rightarrow o}, \Rightarrow_{o\rightarrow o\rightarrow o}, \Leftrightarrow_{o\rightarrow o\rightarrow o}\}$$

$$\cup\, \{\Pi^\alpha_{(\alpha\rightarrow o)\rightarrow o} \mid \alpha \in \mathcal{T}\} \cup \{\Sigma^\alpha_{(\alpha\rightarrow o)\rightarrow o} \mid \alpha \in \mathcal{T}\} \cup \{=^\alpha_{\alpha\rightarrow\alpha\rightarrow o} \mid \alpha \in \mathcal{T}\}.$$

Other constants in a signature are called parameters. The constants Π^α and Σ^α are used to define $\forall$ and $\exists$ (see below) without introducing a binding mechanism other than λ. The set of $\mathcal{HOL}$-formulae (or terms) over Σ are constructed from typed variables and constants using application and λ-abstraction. We let $\mathit{wff}_\alpha(\Sigma)$ be the set of all terms of type α and $\mathit{wff}(\Sigma)$ be the set of all terms. We use $\mathbf{A}, \mathbf{B}, \ldots$ to denote terms in $\mathit{wff}_\alpha(\Sigma)$.

We use vector notation to abbreviate k-fold applications and abstractions as $\mathbf{A}\overline{\mathbf{U}^k}$ and $\lambda\overline{X^k}.\mathbf{A}$, respectively. We also use Church's dot notation so that . stands for a (missing) left bracket whose mate is as far to the right as possible (consistent with given brackets). We use infix notation $\mathbf{A} \vee \mathbf{B}$ for $((\vee\mathbf{A})\mathbf{B})$ and binder notation $\forall X_\alpha.\mathbf{A}$ for $(\Pi^\alpha(\lambda X_\alpha.\mathbf{A}_o))$. While one can consider $\wedge$, $\Rightarrow$ and $\Leftrightarrow$ to be defined (as in [8]), we consider these members of the signature Σ. We also use binder notation $\exists X.\mathbf{A}$ as shorthand for $\Sigma^\alpha(\lambda X.\mathbf{A})$ if Σ^α is a constant in Σ. We let $(\mathbf{A}_\alpha \doteq^\alpha \mathbf{B}_\alpha)$ denote the Leibniz equation $\forall P_{\alpha\to o}.(P\mathbf{A}) \Rightarrow .P\mathbf{B}$.

Each occurrence of a variable in a term is either free or bound by a λ. We use $free(\mathbf{A})$ to denote the set of free variables of $\mathbf{A}$ (i.e., variables with a free occurrence in $\mathbf{A}$). We consider two terms to be equal (written $\mathbf{A} \equiv \mathbf{B}$) if the terms are the same up to the names of bound variables (i.e., we consider α-conversion implicitly). A term $\mathbf{A}$ is closed if $free(\mathbf{A})$ is empty. We let $\mathit{cwff}_\alpha(\Sigma)$ denote the set of closed terms of type α and $\mathit{cwff}(\Sigma)$ denote the set of all closed terms. Each term $\mathbf{A} \in \mathit{wff}_o(\Sigma)$ is called a proposition and each term $\mathbf{A} \in \mathit{cwff}_o(\Sigma)$ is called a sentence.

We denote substitution of a term $\mathbf{A}_\alpha$ for a variable X_α in a term $\mathbf{B}_\beta$ by $[\mathbf{A}/X]\mathbf{B}$. Since we consider α-conversion implicitly, we assume the bound variables of $\mathbf{B}$ avoid variable capture.

Two common relations on terms are given by β-reduction and η-reduction. A β-redex $(\lambda X.\mathbf{A})\mathbf{B}$ β-reduces to $[\mathbf{B}/X]\mathbf{A}$. An η-redex $(\lambda X.\mathbf{C}X)$ (where $X \notin free(\mathbf{C})$) η-reduces to $\mathbf{C}$. For $\mathbf{A}, \mathbf{B} \in \mathit{wff}_\alpha(\Sigma)$, we write $\mathbf{A} \equiv_\beta \mathbf{B}$ to mean $\mathbf{A}$ can be converted to $\mathbf{B}$ by a series of β-reductions and expansions. Similarly, $\mathbf{A} \equiv_{\beta\eta} \mathbf{B}$ means $\mathbf{A}$ can be converted to $\mathbf{B}$ using both β and η. For each $\mathbf{A} \in \mathit{wff}(\Sigma)$ there is a unique β-normal form (denoted $\mathbf{A}{\downarrow_\beta}$) and a unique $\beta\eta$-normal form (denoted $\mathbf{A}{\downarrow_{\beta\eta}}$). From this fact we know $\mathbf{A} \equiv_\beta \mathbf{B}$ ($\mathbf{A} \equiv_{\beta\eta} \mathbf{B}$) iff $\mathbf{A}{\downarrow_\beta} \equiv \mathbf{B}{\downarrow_\beta}$ ($\mathbf{A}{\downarrow_{\beta\eta}} \equiv \mathbf{B}{\downarrow_{\beta\eta}}$).

A non-atomic formula in $\mathit{wff}_o(\Sigma)$ is any formula whose β-normal form is $(c\overline{\mathbf{A}^n})$ where c is a logical constant. An atomic formula is any other formula in $\mathit{wff}_o(\Sigma)$.

Many of the example problems in this paper employ equality, e.g. $\neg(a = \neg a)$. We have different options for the encoding of equality. We can either use primitive equality (i.e., equality as a logical constant) or use some definition of equality in terms of other logical constants. A common definition is Leibniz equality ($\forall P_{\alpha\to o}.(P\mathbf{A}) \Rightarrow .P\mathbf{B}$), but others are possible (see Exercise **X5303** in [4]). In many examples we will denote equality by $\overset{*}{=}$ (e.g., $\neg(a \overset{*}{=} \neg a)$). For each different interpretation of equality, we obtain a different example. We will discuss conditions under which different choices lead to theorems and which choices lead to non-theorems.

For some types, one can also define equality extensionally. For example, one can use equivalence instead of equality at type o. Similarly, at any type $\alpha \to o$, we introduce $\overset{set}{=}$ to denote set equality, i.e., $\overset{set}{=}$ is an abbreviation for

$$\lambda U_{\alpha\to o}\lambda V_{\alpha\to o}\forall X_\alpha.UX \Leftrightarrow VX.$$

In some cases, the use of an extensional definition of equality yields a theorem which can be proven without *assuming* extensionality. We will *not* use the notation $\overset{*}{=}$ to refer to any extensional definition of equality. Interpreting $\overset{*}{=}$ extensionally would significantly change some of the discussion below.

3 Test Problems for Pre-unification and Quantifier Dependencies

Higher-order pre-unification (see [26]) and higher-order Skolemization (see [22]) are important basic ingredients for building an automated higher-order theorem prover. They are largely independent of the chosen semantics for higher-order logic with one exception:β versus $\beta\eta$. As noted in [18] the unification problem relative to β-conversion is different from the unification problem relative to $\beta\eta$-conversion.

3.1 Pre-unification

Implementing a sound, complete and efficient pre-unification algorithm for the simply typed λ-calculus is a highly non-trivial task. Since higher-order pre-unification extends standard first-order unification all first-order test problems in the literature also apply to the higher-order case.

Some specific higher-order test problems can be obtained from the literature on higher-order unification and pre-unification, for example [26,25]. We will now illustrate how further challenging test examples can be easily created using Church numerals.

Church numerals are usually employed in the context of the untyped λ-calculus to encode the natural numbers. This encoding can be partly transformed in a simply typed or polymorphic typed λ-calculus. This includes the definition of successor, addition and multiplication which we employ in or test problems.

Iteration is the key concept to encode natural numbers as Church numerals. For each type α, we can define the Church numeral $\overline{n}^{\alpha}$ by $(\lambda F_{\alpha\to\alpha}\lambda Y_{\alpha}.(F^nY))_{(\alpha\to\alpha)\to(\alpha\to\alpha)}$ where (F^nY) is shorthand for $\underbrace{(F\ (F\ldots(F}_{n-times}\ Y)))$. We will often write $\overline{n}$ instead of $\overline{n}^{\alpha}$, leaving the dependence on the type implicit. Omitting types[1], the successor function $\overline{s}$ can be defined as $\lambda N\lambda F\lambda Y.F(NFY)$, addition $\overline{+}$ as $\lambda M\lambda N\lambda F\lambda Y.MF(NFY)$ and multiplication $\overline{\times}$ as $\lambda M\lambda N\lambda F\lambda Z.N(MF)Z$. To ease notation, we write $\overline{+}$ and $\overline{\times}$ in infix.

Arithmetic equations on Church numerals such as $\overline{3}\overline{\times}\overline{4}\overset{*}{=}\overline{5}\overline{+}\overline{7}$ or $(((\bar{10}\overline{\times}\bar{10})\overline{\times}\bar{10})\overset{*}{=}((\bar{10}\overline{\times}\bar{5})\overline{+}(\bar{5}\overline{\times}\bar{10}))\overline{\times}\bar{10}))$ provide highly suited test problems for the efficiency of β-conversion or $\beta\eta$-conversion in the proof system. Of course, in order to correctly implement β- and η-conversion, one must first properly implement α-conversion.

We obtain more challenging test problems if we employ pre-unification for synthesizing Church numerals and arithmetical operations.

Example 1. (Solving arithmetical equations using pre-unification) The following examples are provable using pre-unification for β-conversion.

[1] N, M are of type $(\alpha\to\alpha)\to(\alpha\to\alpha)$, F is of type $\alpha\to\alpha$, and Y, Z are of type α.

(a) $\exists N_{(\iota\to\iota)\to\iota\to\iota}.((N\overline{\times}\overline{1}) \overset{*}{\doteq} \overline{1})$ (There are two solutions, $\overline{1}$ and $(\lambda F_{\iota\,\iota}.F)$, if one only assumes β-conversion. There is one solution assuming $\beta\eta$-conversion.)
(b) $\exists N.(N\overline{\times}\overline{4}) \overset{*}{\doteq} \overline{5}\overline{+}\overline{7}$
(c) $\exists H.(((H\bar{2})\bar{3}) \overset{*}{\doteq} \bar{6}) \wedge (((H\bar{1})\bar{2}) \overset{*}{\doteq} \bar{2}))$
(d) $\exists N, M.(N\overline{\times}\overline{4}) \overset{*}{\doteq} \overline{5}\overline{+}M$ (There are infinitely many solutions to this problem.)

3.2 Quantifier Dependencies

In proof search with tableaux and expansion proofs, variable conditions can be used to encode quantifier dependencies. Of course, one must be careful to obtain a sound framework. For instance, the variable conditions added with each eliminated existential quantifier in the framework used in [20] allow (incorrect) proofs of the following first-order non-theorems:

Example 2. (First-order non-theorems)

(a) (Example 2.9 in [30]) $(\exists X_\iota \forall Y_\iota . q_{\iota\to\iota\to o} XY) \vee (\exists U_\iota \forall V_\iota . \neg q V U)$
(b) (Example 2.50 in [30]) $\exists Y_\iota \forall X_\iota . ((\forall Z_\iota . q_{\iota\to\iota\to o} XZ) \vee (\neg q XY))$

In [19] an attempt was made to use variable conditions in the context of resolution theorem proving (for a sorted extension of higher-order logic) instead of introducing Skolem terms. However, the system was unsound as it allowed a resolution refutation proving the following non-theorem:

Example 3. (Non-Theorem: Every function has a fixed point) $\forall F_{\alpha\to\alpha} . \exists X_\alpha . F\ X \doteq X$. The idea is that one obtains two single-literal clauses $(P_{\iota\to o}(FX))$ and $\neg(PY)$ using clause normalization and variable renaming (where X and Y can be instantiated). One then obtains the empty clause by unifying Y with (FX).

Skolem terms avoid incorrect proofs of such theorems since the Skolem terms will preserve the relationship between renamed variables in different clauses. In particular, if S is a Skolem function, we would obtain single-literal clauses $(S_{\iota\to\iota\to o} X(FX))$ and $\neg(S_{\iota\to\iota\to o} YY)$ which cannot be resolved and unified.

There is a relationship between Skolemization and the axiom of choice in the first-order case which becomes more delicate in the higher-order case. Consider formulas $\forall x_\iota \exists y_\iota \varphi(x, y)$ and $\forall x_\iota \varphi(x, (f_{\iota\to\iota} x))$. In first-order logic, the two formulas are equivalent with respect to satisfiability whenever f does not occur in φ. The equivalence follows from the fact that any first-order model (with domain $\mathcal{D}_\iota$) satisfying $\forall x \exists y \varphi(x, y)$ can be extended to interpret f as a function $g : \mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$ such that $\forall x \varphi(x, (fx))$ holds. In general, the axiom of choice (at the *meta-level*) is required to conclude the function g exists. The situation is different in the higher-order case. As we shall see when we consider higher-order models, we would need to interpret f not simply as a function from $\mathcal{D}_\iota$ to $\mathcal{D}_\iota$, but as a member of a domain $\mathcal{D}_{\iota\to\iota}$. Existence of an appropriate function from $\mathcal{D}_\iota$ to $\mathcal{D}_\iota$ follows from the axiom of choice at the meta-level, but the existence of an appropriate element of $\mathcal{D}_{\iota\to\iota}$ would only follow from a choice property internal to the higher-order model.

Dale Miller has shown that a naive adaptation of standard first-order Skolemization to higher-order logic allows one to prove particular instances of the axiom of choice.

For example, naive Skolemization permits an easy proof of the following version of the axiom of choice:

Example 4. (Choice) $(\forall X \exists Y \mathbf{.} rXY) \Rightarrow (\exists F \forall X \mathbf{.} rX(FX))$

However, naive Skolemization does not provide a complete method for reasoning with choice. The following example is equivalent to the axiom of choice (essentially Axiom 11 in [15]) but is not provable using naive Skolemization.

Example 5. (Choice) $\exists E_{(\iota \rightarrow o) \rightarrow \iota} \forall P \mathbf{.} (\exists Y \mathbf{.} PY) \Rightarrow \mathbf{.} P(EP)$

Thus standard first-order Skolemization is unsound in higher-order logic as it partly introduces choice into the proof system. Dale Miller has fixed the problem by adding further conditions (see [22]): any Skolem function symbol f^n with dependency arity n (the existentially bound variable to be eliminated by a new Skolem term headed by f is depending on n universal variables) may only occur in formulas $f^n \overline{\mathbf{A}^n}$, where none of the $\mathbf{A}^i$ contains a variable that is bound outside of the term $f^n \overline{\mathbf{A}^n}$.

4 Semantics for HOL

In [8] we have re-examined the semantics of classical higher-order logic with the purpose of clarifying the role of extensionality. For this we have defined eight classes of higher-order models with respect to various combinations of Boolean extensionality and three forms of functional extensionality. One can further refine these eight model classes by varying the logical constants in the signature Σ as in [14].

A model of $\mathcal{HOL}$ is given by four objects: a typed collection of nonempty sets $(\mathcal{D}_\alpha)_{\alpha \in \mathcal{T}}$, an application operator $@ \colon \mathcal{D}_{\alpha \rightarrow \beta} \times \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$, an evaluation function $\mathcal{E}$ for terms and a valuation function $v \colon \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$. A pair $(\mathcal{D}, @)$ is called a Σ-applicative structure (see [8](3.1)). If $\mathcal{E}$ is an evaluation function for $(\mathcal{D}, @)$ (see [8](3.18)), then we call the triple $(\mathcal{D}, @, \mathcal{E})$ a Σ-evaluation. If v satisfies appropriate properties, then we call the tuple $(\mathcal{D}, @, \mathcal{E}, v)$ a Σ-model (see [8](3.40 and 3.41)).

Given an applicative structure $(\mathcal{D}, @)$, an assignment φ is a (typed) function from $\mathcal{V}$ to $\mathcal{D}$. An evaluation function $\mathcal{E}$ maps an assignment φ and a term $\mathbf{A}_\alpha \in \mathit{wff}_\alpha(\Sigma)$ to an element $\mathcal{E}_\varphi(\mathbf{A}) \in \mathcal{D}_\alpha$. Evaluation functions $\mathcal{E}$ are required to satisfy four properties given in [8](3.18)). If $\mathbf{A}$ is closed and $\mathcal{E}$ is an evaluation function, then $\mathcal{E}_\varphi(\mathbf{A})$ cannot depend on φ and we write $\mathcal{E}(\mathbf{A})$.

A valuation $v \colon \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ is required to satisfy a property $\mathfrak{L}_c(\mathcal{E}(c))$ for every logical constant $c \in \Sigma$ (see [8](3.40)). For each logical constant c, $\mathfrak{L}_c(\mathsf{a})$ is defined to hold if a is an object of a domain $\mathcal{D}_\alpha$ satisfying the characterizing property of the logical constant c. For example, $\mathfrak{L}_\neg(\mathsf{n})$ holds for $\mathsf{n} \in \mathcal{D}_{o \rightarrow o}$ iff for every $\mathsf{a} \in \mathcal{D}_o$, $v(\mathsf{n}@\mathsf{a})$ is $\mathtt{T}$ iff $v(\mathsf{a})$ is $\mathtt{F}$. Likewise, $\mathfrak{L}_{=^\alpha}(\mathsf{q})$ holds for $\mathsf{q} \in \mathcal{D}_{\alpha \rightarrow \alpha \rightarrow o}$ if for every $\mathsf{a}, \mathsf{b} \in \mathcal{D}_\alpha$, $v(\mathsf{q}@\mathsf{a}@\mathsf{b})$ is $\mathtt{T}$ iff a equals b.

Given a model $\mathcal{M} := (\mathcal{D}, @, \mathcal{E}, v)$, an assignment φ and a proposition $\mathbf{A}$ (or set of propositions Φ), we say $\mathcal{M}$ satisfies $\mathbf{A}$ (or Φ) and write $\mathcal{M} \models_\varphi \mathbf{A}$ (or $\mathcal{M} \models_\varphi \Phi$) if $v(\mathcal{E}_\varphi(\mathbf{A})) \equiv \mathtt{T}$ (or $v(\mathcal{E}_\varphi(\mathbf{A})) \equiv \mathtt{T}$ for each $\mathbf{A} \in \Phi$). If $\mathbf{A}$ is closed (or every member of Φ is closed), then we simply write $\mathcal{M} \models \mathbf{A}$ (or $\mathcal{M} \models \Phi$) and say $\mathcal{M}$ is a model of $\mathbf{A}$ (or Φ). We also consider classes $\mathfrak{M}$ of Σ-models and say a proposition $\mathbf{A}$ is valid in $\mathfrak{M}$ if $\mathcal{M} \models_\varphi \mathbf{A}$ for every $\mathcal{M} \in \mathfrak{M}$ and assignment φ.

In order to define model classes which correspond to different notions of extensionality, we define five properties of models (see [8](3.46, 3.21 and 3.5)). For each Σ-model $\mathcal{M} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$, we say $\mathcal{M}$ satisfies property

$\mathfrak{q}$ iff for all $\alpha \in \mathcal{T}$ there is a $\mathsf{q}^\alpha \in \mathcal{D}_{\alpha\to\alpha\to o}$ with $\mathfrak{L}_{=^\alpha}(\mathsf{q}^\alpha)$.
η iff $(\mathcal{D}, @, \mathcal{E})$ is η-functional (i.e., for each $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma)$ and assignment φ, $\mathcal{E}_\varphi(\mathbf{A}) \equiv \mathcal{E}_\varphi(\mathbf{A}{\downarrow_{\beta\eta}})$).
ξ iff $(\mathcal{D}, @, \mathcal{E})$ is ξ-functional (i.e., for each $\mathbf{M}, \mathbf{N} \in \mathit{wff}_\beta(\Sigma)$, $X \in \mathcal{V}_\alpha$ and assignment φ, $\mathcal{E}_\varphi(\lambda X_\alpha . \mathbf{M}_\beta) \equiv \mathcal{E}_\varphi(\lambda X_\alpha . \mathbf{N}_\beta)$ whenever $\mathcal{E}_{\varphi,[\mathsf{a}/X]}(\mathbf{M}) \equiv \mathcal{E}_{\varphi,[\mathsf{a}/X]}(\mathbf{N})$ for every $\mathsf{a} \in \mathcal{D}_\alpha$).
$\mathfrak{f}$ iff $(\mathcal{D}, @)$ is functional (i.e., for each $\mathsf{f}, \mathsf{g} \in \mathcal{D}_{\alpha\to\beta}$, $\mathsf{f} \equiv \mathsf{g}$ whenever $\mathsf{f}@\mathsf{a} \equiv \mathsf{g}@\mathsf{a}$ for every $\mathsf{a} \in \mathcal{D}_\alpha$).
$\mathfrak{b}$ iff υ is injective.

For each $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$ and each signature Σ we define $\mathfrak{M}_*(\Sigma)$ to be the class of all Σ-models $\mathcal{M}$ such that $\mathcal{M}$ satisfies property $\mathfrak{q}$ and each of the additional properties $\{\eta, \xi, \mathfrak{f}, \mathfrak{b}\}$ indicated in the subscript $*$ (see [8](3.49)). We always include β in the subscript to indicate that β-equal terms are always interpreted as identical elements. We do not include property $\mathfrak{q}$ as an explicit subscript; $\mathfrak{q}$ is treated as a basic, implicit requirement for all model classes. See [8](3.52) for a discussion on why we require property $\mathfrak{q}$. (We also briefly explore models which do not satisfy property $\mathfrak{q}$ in the context of Example 8 and again in Subsection 5.3.) Since we are varying four properties, one would expect to obtain 16 model classes. However, we showed in [8] that $\mathfrak{f}$ is equivalent to the conjunction of ξ and η. Note that, for example, $\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma)$ is a larger class of models than $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$, hence fewer propositions are valid in $\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma)$ than are valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$. In our examples we try to indicate the largest of our model classes in which the proposition is valid. Implicitly, this means the proposition is also valid in smaller (more restricted) model classes and may not be valid in larger (less restricted) ones.

5 Test Problems for Higher-Order Theories

Unless stated otherwise, we assume the signature includes $\overline{\Sigma}$ (see p. 67) and write $\mathfrak{M}_*$ for $\mathfrak{M}_*(\Sigma)$. Many of the examples could be considered in the context of smaller signatures. In the following discussion, we only consider smaller signatures in order to make particular points. (Note that if the signature becomes too small, Leibniz equality, for example, is no longer expressible.)

5.1 Properties of Equality

There are many useful first-order test problems on equality reasoning in the literature. For instance, in [12] the following clause set is given to illustrate the incompleteness of the RUE-NRF resolution approach as introduced in [16]:

$$\{g(f(a)) = a, f(g(X)) \neq X\}$$

Here, X is a free variable (i.e., implicitly universially quantified) and f, g are unary function symbols. In [12] it is shown that this inconsistent clause set cannot be refuted in the first-order RUE-NRF approach.

We now present some higher-order test problems addressing properties of equality. Some of them apply to many possible notions of equality while others describe specific properties of individual notions or relate different notions to each other.

Example 6. Equality is an equivalence relation in $\mathfrak{M}_\beta$. These particular examples should be theorems even if one replaces $\overset{*}{=}$ with an extensional definition of equality (e.g., $\Leftrightarrow$ at type o or $\overset{set}{=}$ at any type $\alpha \to o$).

(a) $\forall X_\alpha . X \overset{*}{=} X$
(b) $\forall X_\alpha \forall Y_\alpha . X \overset{*}{=} Y \Rightarrow Y \overset{*}{=} X$
(c) $\forall X_\alpha \forall Y_\alpha \forall Z_\alpha . (X \overset{*}{=} Y \land Y \overset{*}{=} Z) \Rightarrow X \overset{*}{=} Z$

Example 7. Equality obeys the congruence property (substitutivity property) in $\mathfrak{M}_\beta$.

(a) $\forall X_\alpha \forall Y_\alpha \forall F_{\alpha \to \alpha} . X \overset{*}{=} Y \Rightarrow (FX) \overset{*}{=} (FY)$
(b) $\forall X_\alpha \forall Y_\alpha \forall P_{\alpha \to o} . (X \overset{*}{=} Y) \land (PX) \Rightarrow (PY)$

Example 8 relates the Leibniz definition of equality to primitive equality.

Example 8. $(a_\alpha \dot{=}^\alpha b_\alpha) \Rightarrow (a =^\alpha b)$.

One could legitimately debate whether Example 8 should be a theorem. On the one hand, if Example 8 is not a theorem, then one should not consider Leibniz equality to be a definition of *real* equality. Semantically, Henkin's first (quite natural) definitions of models allowed models in which Leibniz equality (e.g., at type ι) does *not* evaluate to equality of objects in the model. Such a model $\mathcal{M}$ is constructed in [1]. This model $\mathcal{M}$ is a Σ-model in the sense of this paper (if one assumes $=^\alpha \notin \Sigma$ for every type α), but is not in any model class $\mathfrak{M}_*(\Sigma)$ since property $\mathfrak{q}$ fails. There is a slight technical problem with saying $\mathcal{M}$ provides a counter-model for Example 8 since one cannot express Example 8 without $=^\iota \in \Sigma$. As in [14], one can distinguish between *internal* and *external* uses of equality (as well as $\Rightarrow$ and $\forall$) and determine that $\mathcal{M}$ *is* (in a sense that can be made precise) a countermodel for Example 8.

If a model satisfies property $\mathfrak{q}$, then Example 8 is valid for any type α. If a logical system is intended to be complete for one of our model classes $\mathfrak{M}_*(\Sigma)$, then Example 8 should be a theorem. For the complete natural deduction calculi in [8], there is an explicit rule which derives primitive equality from Leibniz equality. In some sense, requiring property $\mathfrak{q}$ semantically corresponds to explicitly requiring that Example 8 be provable.

Also, if $=^\alpha \in \Sigma$, then Example 8 (for this particular type α) is valid in any Σ-model. A proof *using* primitive equality could instantiate the Leibniz variable $P_{\alpha \to o}$ with $(\lambda Z_\alpha . a = Z)$. The important point is that $=$ must be available for instantiations during proofs (not simply for expressing the original sentence).

Extensionality is the distinguishing property motivating our different model classes. For both, functional and Boolean extensionality, we distinguish between a trivial and a non-trivial direction.

Example 9. The trivial directions of functional and Boolean extensionality are valid in $\mathfrak{M}_\beta$.

(a) $\forall F_{\alpha\to\beta}\forall G_{\alpha\to\beta}.F \overset{*}{=} G \Rightarrow (\forall X_\alpha.(FX) \overset{*}{=} (GX))$
(b) $\forall A_o \forall B_o.A \overset{*}{=} B \Rightarrow (A \Leftrightarrow B)$

The other directions are not valid in $\mathfrak{M}_\beta$. They become theorems only relative to more restricted model classes in our landscape.

Example 10. (discussed in [15]; Axiom 10 in [17]) $\forall A_o \forall B_o.(A \Leftrightarrow B) \Rightarrow A \overset{*}{=} B$ is valid in $\mathfrak{M}_{\beta\mathfrak{b}}$. This is the non-trivial direction of Boolean extensionality.

Example 11. ([15,17], Axiom $10^{\beta\alpha}$) $\forall F_{\alpha\to\beta}\forall G_{\alpha\to\beta}.(\forall X_\alpha.(FX) \overset{*}{=} (GX)) \Rightarrow F \overset{*}{=} G$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}}$. This is the non-trivial direction of functional extensionality. (Property $\mathfrak{q}$ is also relevant to this example as is discussed in [8].)

5.2 Extensionality

We next present examples that illustrate distinguishing properties of the different model classes with respect to extensionality. In the preceding sections we have already mentioned several test problems that are independent of the "amount of extensionality" and which are theorems in $\mathfrak{M}_\beta$. We additionally refer to all first-order test problems as, for instance, provided in the TPTP library.

η-equality is usually realized as part of the pre-unification algorithm in a higher-order reasoning system. It is important to note that η-equality should not be confused with full extensionality. In literature on higher-order rewriting, for instance [25], the notion of extensionality is usually only associated with η-conversion which is far less than full extensionality.

Example 12. $(p_{(\iota\to\iota)\to o}(\lambda X_\iota.f_{\iota\to\iota}X))\Rightarrow(p_{(\iota\to\iota)\to o}f)$ is essentially 21 from [15] which expresses η-equality using Leibniz equality. It is valid in $\mathfrak{M}_{\beta\eta}$ but not in $\mathfrak{M}_\beta$.

Property ξ together with η gives us full functional extensionality.

Example 13. Validity of $(\forall X_\iota.(f_{\iota\to\iota}X) \overset{*}{=} X) \wedge p(\lambda X_\iota X) \Rightarrow p(\lambda X_\iota.fX)$ only depends on ξ, not on η. It is thus valid in $\mathfrak{M}_{\beta\xi}$ (but not in model classes which do not require either ξ or $\mathfrak{f}$).

Example 14. $(\forall X_\iota.(f_{\iota\to\iota}X) \overset{*}{=} X) \wedge p(\lambda X_\iota X) \Rightarrow pf$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}}$, but not in model classes which do not require $\mathfrak{f}$.

As in Example 11, property $\mathfrak{q}$ is important for validity of Example 13 in $\mathfrak{M}_{\beta\xi}$ and validity of Example 14 in $\mathfrak{M}_{\beta\mathfrak{f}}$.

Example 15. ([7]) (a) $p_{o\to o}\ (a_o \wedge b_o) \Rightarrow p\ (b \wedge a)$ and (b) $a_o \wedge b_o \wedge (p_{o\to o}a) \Rightarrow (pb)$ are valid iff we require Boolean extensionality as in $\mathfrak{M}_{\beta\mathfrak{b}}$.

Example 16. $(p_{o\to o}\ a_o) \wedge (p\ b_o) \Rightarrow (p\ (a \wedge b))$ is a theorem of $\mathfrak{M}_{\beta\mathfrak{b}}$ which is slightly more complicated to mechanize in some calculi; see [7] for more details.

Example 17. $\neg(a = \neg a)$ is valid in $\mathfrak{M}_{\beta\mathfrak{b}}$. As discussed in [7] this example motivates specific inference rules for the mechanization of primitive equality.

The following is a tricky example introduced in [14].

Example 18. $(h_{o\to\iota}((h\top) \overset{*}{=} (h\bot))) \overset{*}{=} (h\bot)$ is valid in $\mathfrak{M}_{\beta\mathfrak{b}}$, but not in model classes which do not require property $\mathfrak{b}$.

Many people do not immediately accept that Example 18 is a theorem. A simple informal argument is helpful. Either $(h\top) \overset{*}{=} (h\bot)$ is true or false. If the equation holds, then Example 18 reduces to $(h\top) \overset{*}{=} (h\bot)$ which we have just assumed. If the equation is false, then Example 18 reduces to $(h\bot) \overset{*}{=} (h\bot)$, an instance of reflexivity.

Example 19 combines Boolean extensionality with η-equality.

Example 19. $p_{(\iota\to\iota)\to o}(\lambda X_\iota.f_{o\to\iota\to\iota}(a_{(\iota\to\iota)\to o}(\lambda X_\iota.fb_oX)\wedge b)X) \Rightarrow p(f(b\wedge.a(fb)))$ is valid in $\mathfrak{M}_{\beta\eta\mathfrak{b}}$, but is not valid if properties $\mathfrak{b}$ and η are not assumed.

By DeMorgan's Law, we know $X \wedge Y$ is the same as $\neg(\neg X \vee \neg Y)$. In Example 20, we vary the notion of "is the same as" to obtain several examples which are only provable with some amount of extensionality. Note that if we only assume property ξ, we can only conclude the η-expanded form of $\wedge$ is equal to $(\lambda X \lambda Y.\neg(\neg X \vee \neg Y))$.

Example 20. Consider the following examples.

(a) $\forall X \forall Y.X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$ is valid in $\mathfrak{M}_\beta$.
(b) $\forall X \forall Y.X \wedge Y \overset{*}{=} \neg(\neg X \vee \neg Y)$ is valid in $\mathfrak{M}_{\beta\mathfrak{b}}$.
(c) $(\lambda U \lambda V.U \wedge V) \overset{*}{=} (\lambda X \lambda Y.\neg(\neg X \vee \neg Y))$ is valid in $\mathfrak{M}_{\beta\xi\mathfrak{b}}$.
(d) $\wedge \overset{*}{=} (\lambda X \lambda Y.\neg(\neg X \vee \neg Y))$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$.

Finally we reach Henkin semantics which is characterized by full extensionality, i.e. the combination of Boolean and functional extensionality. Example 20(d) already provided one example valid only in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$.

Example 21. The following theorem in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ characterizes the fact that in all Henkin models we have exactly four functions mapping truth values to truth values.

$$((p\ \lambda X_o.X_o) \wedge (p\ \lambda X_o.\neg X_o) \wedge (p\ \lambda X_o.\bot) \wedge (p\ \lambda X_o.\top)) \Rightarrow \forall Y_{o\to o}.(p\ Y)$$

Example 22. As exploited in [11], set theory problems can be concisely and elegantly formulated in higher-order logic when using λ-abstraction to encode sets as characteristic functions. For instance, given a predicate $p_{\alpha\to o}$ the set of all objects of type α that have property p is denoted as $\lambda X_\alpha.(pX)$. We then define set operations as follows (we give only some examples):

set operation	defined by
$\in_{\alpha\to(\alpha\to o)\to o}$	$\lambda Z_\alpha \lambda X_{\alpha\to o}(XZ)$
$\{\cdot\}_{\alpha\to(\alpha\to o)}$	$\lambda U_\alpha(\lambda Z_\alpha.Z \overset{*}{=} U)$
$\emptyset_{\alpha\to o}$	$(\lambda Z_\alpha \bot)$
$\cap_{(\alpha\to o)\to(\alpha\to o)\to(\alpha\to o)}$	$\lambda X_{\alpha\to o}\lambda Y_{\alpha\to o}(\lambda Z_\alpha.Z \in X \wedge Y \in Y)$
$\cup_{(\alpha\to o)\to(\alpha\to o)\to(\alpha\to o)}$	$\lambda X_{\alpha\to o}\lambda Y_{\alpha\to o}(\lambda Z_\alpha.Z \in X \vee Y \in Y)$
$\subseteq_{(\alpha\to o)\to(\alpha\to o)\to o}$	$\lambda X_{\alpha\to o}\lambda Y_{\alpha\to o}(\forall Z_\alpha.Z \in X \Rightarrow Y \in Y)$
$\wp_{(\alpha\to o)\to((\alpha\to o)\to o)}$	$\lambda X_{\alpha\to o}(\lambda Y_{\alpha\to o}.Y \subseteq X)$

We can now formulate some test problems on sets:

(a) $a_{\alpha\to o} \cup (b_{\alpha\to o} \cap c_{\alpha\to o}) \overset{set}{=} (a \cup b) \cap (a \cup c)$ is valid in $\mathfrak{M}_\beta$.
(b) $a_{\alpha\to o} \cup (b_{\alpha\to o} \cap c_{\alpha\to o}) \overset{*}{=} (a \cup b) \cap (a \cup c)$ is valid in $\mathfrak{M}_{\beta\xi\mathfrak{b}}$ but not in model classes without ξ and $\mathfrak{b}$.
(c) $\wp(\emptyset_{\alpha\to o}) \overset{set}{=} \{\emptyset_{\alpha\to o}\}$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ but not in model classes without $\mathfrak{f}$ and $\mathfrak{b}$. The example is not valid in $\mathfrak{M}_\beta$ due to the embedded equation introduced by the definition of a singleton set $\{.\}$.
(d) and $\wp(\emptyset_{\alpha\to o}) \overset{*}{=} \{\emptyset_{\alpha\to o}\}$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ but not in model classes without $\mathfrak{f}$ and $\mathfrak{b}$.

These examples motivate pre-processing in higher-order theorem proving in which the definitions are fully expanded and in which the extensionality principles are employed es early as possible. After pre-processing, many problems of this kind can be automatically translated from their concise and human readable higher-order representation into first-order or even propositional logic representations to be easily checked by respective specialist systems.

5.3 Set Comprehension

One of the advantages of Church's type theory is that instead of assuming comprehension axioms one can simply use terms defining sets for set instantiations. Such set instantiations make use of logical constants in the signature Σ. As in [14] one can vary the signature of logical constants in order to vary the set comprehension assumed in Σ-models. With different amounts of set comprehension, different examples will be valid.

Generating set instantiations is one of the toughest challenges for the automation of higher-order logic. (In fact set instantiations can be employed to simulate the cut-rule as soon as one of the following prominent axioms of higher-order logic is available in the search space: comprehension, induction, extensionality, choice, description.) Set instantiations are often generated during automated search using an enumeration technique involving primitive substitutions.

For each example below, we note restrictions on the signature Σ under which the example is either valid or not valid. Since we would like to distinguish between signatures which contain primitive equality (at various types) and those which do not, we consider classes of models which do not necessarily satisfy property $\mathfrak{q}$. In particular, let $\mathfrak{M}_\beta^{-\mathfrak{q}}(\Sigma)$ be the set of all Σ-models and let $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}^{-\mathfrak{q}}(\Sigma)$ be the set of all Σ-models satisfying properties $\mathfrak{f}$ and $\mathfrak{b}$ (without requiring property $\mathfrak{q}$).

As in Example 8 one can focus on the use of logical constants in Σ for instantiations and ignore certain uses of logical constants to express the formula. For example, suppose $\mathbf{A} \in cwff_o(\Sigma)$, $\mathcal{M}$ is a Σ-model and $\neg \notin \Sigma$. While $(\neg\mathbf{A}) \notin wff_o(\Sigma)$, we can consider $(\neg\mathbf{A})$ to be a *Σ-external proposition* and define $\mathcal{M} \models \neg\mathbf{A}$ to mean $\mathcal{M} \not\models \mathbf{A}$. Intuitively, the negation is used *externally* in $(\neg\mathbf{A})$. We can inductively define the set of Σ-external propositions $\mathbf{M}$ and the meaning of $\mathcal{M} \models \mathbf{M}$ for Σ-models $\mathcal{M}$. After doing so, most of the examples below are Σ-external propositions even if Σ contains no logical constants. Only Examples 30 and 33 in this section make nontrivial uses of certain logical constants to express the propositions. Due to space considerations, we refer the reader to [14] for details.

If Σ is sufficiently small, then one can construct two trivial models in $\mathfrak{M}_{\beta\mathfrak{fb}}^{-\mathfrak{q}}(\Sigma)$ where $\mathcal{D}_o$ is either simply $\{\mathtt{T}\}$ or $\{\mathtt{F}\}$. (This possibility was ruled out in [8] since we assumed $\neg \in \Sigma$.)

Example 23. $\exists PP$ is valid in $\mathfrak{M}_{\beta}^{-\mathfrak{q}}(\Sigma)$ if either $\top \in \Sigma$ or $\neg \in \Sigma$. The example is not valid in $\mathfrak{M}_{\beta\mathfrak{fb}}^{-\mathfrak{q}}(\Sigma)$ if $\Sigma \subseteq \{\bot, \wedge, \vee\} \cup \{\Pi^\alpha, \Sigma^\alpha | \alpha \in \mathcal{T}\}$. (Any proof must use a set instantiation involving either $\top$, $\neg$, $\Rightarrow$, $\Leftrightarrow$ or some primitive equality.)

Example 24. $\neg\forall PP$ is valid in $\mathfrak{M}_{\beta}^{-\mathfrak{q}}(\Sigma)$ if either $\bot \in \Sigma$ or $\neg \in \Sigma$. The example is not valid in $\mathfrak{M}_{\beta\mathfrak{fb}}^{-\mathfrak{q}}(\Sigma)$ if $\Sigma \subseteq (\overline{\Sigma} \setminus \{\bot, \neg\})$. (Any proof must use a set instantiation involving either $\bot$ or $\neg$.)

Example 25 characterizes when an instantiation satisfying the property of negation is possible. This can be either because the signature supplies negation or supplies enough constants to define negation.

Example 25. $\exists N_{o\to o}\forall P_o.NP \Leftrightarrow \neg P$ is valid in $\mathfrak{M}_{\beta}^{-\mathfrak{q}}(\Sigma)$ if $\neg \in \Sigma$. The example is also valid in $\mathfrak{M}_{\beta}^{-\mathfrak{q}}(\Sigma)$ if $\bot \in \Sigma$ and $\{\Rightarrow, \Leftrightarrow\} \cap \Sigma \neq \emptyset$ since one can consider either the term $\lambda X_o.X \Rightarrow \bot$ or the term $\lambda X_o.X \Leftrightarrow \bot$. The example is not valid in $\mathfrak{M}_{\beta\mathfrak{fb}}^{-\mathfrak{q}}(\Sigma)$ if $\Sigma \subseteq \{\top, \bot, \wedge, \vee\} \cup \{\Pi^\alpha, \Sigma^\alpha | \alpha \in \mathcal{T}\}$.

One possibility we did not cover in Example 25 is if Σ is $\{\bot, =^o\}$. Consider the term $(\lambda X_o.X =^o \bot)$. This only defines negation if we assume Boolean extensionality. Hence we obtain the interesting fact that Example 25 is valid in $\mathfrak{M}_{\beta\mathfrak{fb}}^{-\mathfrak{q}}(\{\bot, =^o\})$, but is *not* valid in $\mathfrak{M}_{\beta}^{-\mathfrak{q}}(\{\bot, =^o\})$.

One can modify Example 25 in a way that requires not only a set instantiation for negation, but also extensionality.

Example 26. $\neg\forall F_{o\to o}\exists X.(FX) \stackrel{*}{=} X$ is valid in $\mathfrak{M}_{\beta\mathfrak{fb}}^{-\mathfrak{q}}(\Sigma)$ if $\neg \in \Sigma$. The example is not valid in $\mathfrak{M}_{\beta}^{-\mathfrak{q}}(\Sigma)$ regardless of the signature Σ. Also, the example is not valid in $\mathfrak{M}_{\beta\mathfrak{fb}}^{-\mathfrak{q}}(\Sigma)$ if $\Sigma \subseteq \{\top, \bot, \wedge, \vee\} \cup \{\Pi^\alpha, \Sigma^\alpha | \alpha \in \mathcal{T}\}$.

Example 27 characterizes when an instantiation can essentially define disjunction and Example 28 characterizes when an instantiation can essentially define the universal quantifier at type α. Clearly one can modify these examples for any other logical constant.

Example 27. $\exists D_{o\to o\to o}\forall P_o\forall Q_o.DPQ \Leftrightarrow (P \vee Q)$ is valid in $\mathfrak{M}_{\beta}^{-\mathfrak{q}}(\Sigma)$ if $\vee \in \Sigma$. The example is also valid in $\mathfrak{M}_{\beta}^{-\mathfrak{q}}(\Sigma)$ if $\{\neg, \wedge\} \subseteq \Sigma$.

Example 28. $\exists Q_{(\alpha\to o)\to o)}\forall P_{\alpha\to o}.QP \Leftrightarrow \forall X_\alpha.PX$ is valid in $\mathfrak{M}_{\beta}^{-\mathfrak{q}}(\Sigma)$ if $\Pi^\alpha \in \Sigma$.

Recall that Example 8 already provided an example in which one *might* require a set instantiation involving primitive equality (depending on how the calculus relates Leibniz equality to primitive equality).

A few interesting set instantiations involve no logical constants, but do make use of projections (see [18]). Sometimes such projections can be obtained from higher-order unification, as in Example 29.

Example 29. $\exists N_{o\to o} \forall P_o . NP \Leftrightarrow P$ is valid in $\mathfrak{M}_\beta^{-\mathfrak{q}}(\emptyset)$.

However, one cannot expect higher-order unification to *always* provide projection terms when they are needed. Example 30 was studied extensively in [2] (see THM104) in order to demonstrate this fact. In this example, we make use of the abbreviation $\{.\}$ which was defined in Example 22. If the definition of $\{.\}$ makes use of primitive equality, one must assume $=^\iota \in \Sigma$ to express the proposition. If $\{.\}$ is defined using Leibniz equality, then one must assume $\neg, \Pi^{\iota\to o} \in \Sigma$ to express the proposition.

Example 30. $\forall X_\iota \forall Z_\iota . \{X\} \doteq \{Z\} \Rightarrow X \doteq Z$ is valid in $\mathfrak{M}_\beta^{-\mathfrak{q}}(\Sigma)$ so long as Σ is sufficient to express the proposition.

The examples above are straightforward examples designed to ensure completeness of theorem provers with respect to set comprehension. A more natural theorem which requires set instantiations is Cantor's Theorem. Two forms of Cantor's Theorem were studied with respect to set comprehension in [14]. Example 31 is the surjective form of Cantor's Theorem discussed in [4].

Example 31. (Surjective Cantor Theorem) $\neg \exists G_{\alpha\to\alpha\to o} \forall F_{\alpha\to o} \exists J_\alpha . GJ =^{\alpha\to o} F$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}^{-\mathfrak{q}}(\Sigma)$ if $\neg \in \Sigma$. The example is not valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}^{-\mathfrak{q}}(\Sigma)$ if $\Sigma \subseteq \{\top, \bot, \wedge, \vee\} \cup \{\Pi^\alpha, \Sigma^\alpha | \alpha \in \mathcal{T}\}$ (see Theorem 6.7.8 in [14]).

An alternative formulation of Cantor's Theorem (see [5,14]) is the injective form shown in Example 32. Almost any higher-order theorem prover complete for the corresponding model class should be capable of proving the previous examples in this subsection. Example 32 is far more challenging. At the present time, no theorem prover has found a proof of Example 32 automatically.

Example 32. (Injective Cantor Theorem) $\neg \exists H_{(\iota\to o)\to\iota} \forall P_{\iota\to o} \forall Q_{\iota\to o} . HP =^\iota HQ \Rightarrow P =^{\iota\to o} Q$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}^{-\mathfrak{q}}(\Sigma)$ if $\{\neg, \wedge, =^\iota, \Pi^{\iota\to o}\} \subseteq \Sigma$ (see Lemma 6.7.2 in [14]). The example is not valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}^{-\mathfrak{q}}(\Sigma)$ if $\Sigma \subseteq \{\top, \bot, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \Pi^\iota, \Sigma^\iota, =^{\iota\to o}\}$. (This fact follows from the results in Section 6.7 of [14].)

One of the difficulties of proving Example 32 is that certain set instantiations seem to be needed beneath other set instantiations (see [5]). The next family of examples illustrates that nontrivial set instantiations can occur within set instantiations with an arbitrary number of iterations.

Example 33. Assume Σ contains $\neg$ and Π^α for every type α. Fix a constant c_ι. We will define a theorem $\mathbf{D}_o^n$ for each natural number n. By induction on n, define simple types τ^n and abbreviations $\mathbf{A}_{\tau^n\to o}^n$ as follows.

(a) Let τ^0 be the type ι and τ^{n+1} be $\tau^n \to o$ for each natural number n.
(b) Let $\mathbf{A}_{\iota\to o}^0$ be $\lambda Z . (Z \doteq c_\iota) \wedge \top$ and $\mathbf{A}^{n+1}$ be $\lambda Z_{\tau^{n+1}} . (Z \doteq \mathbf{A}^n) \wedge \exists T_{\tau^n} . ZT$ for each natural number n.

Finally, for each n, let $\mathbf{D}_o^n$ be $\exists S_{\tau^n} A^n S$. Each $\mathbf{D}^n$ is a valid in $\mathfrak{M}_\beta^{-\mathfrak{q}}(\Sigma)$. The constant c_ι is the obvious witness for $\mathbf{D}^0$. For each n, $\mathbf{A}^n$ is the witness for $\mathbf{D}^{n+1}$. Note that a subgoal of showing $\mathbf{A}^n$ is the witness for $\mathbf{D}^{n+1}$ involves showing $\mathbf{A}^n$ is nonempty (which was $\mathbf{D}^n$). Hence this proof of $\mathbf{D}^{n+1}$ involves all the previous instantiations $\mathbf{A}^0, \ldots, \mathbf{A}^n$.

6 More Complex Examples

Here we present technically or proof theoretically challenging examples. First we consider a class of hard problems simply involving β-reduction.

Example 34. Let α^0 be ι and α^{n+1} be $(\alpha^n \to \alpha^n)$ for each n. Note that the Church numeral $\overline{2}^{\alpha^n}$ has type α^{n+2}. For any n we can form the term $(\overline{2}^{\alpha^n}\overline{2}^{\alpha^{n-1}}\cdots\overline{2}^{\alpha^0})$ of type $(\iota \to \iota) \to \iota \to \iota$. The size of the β-normal form of this term is approximately of size $2^{(2^{\cdot^{\cdot^{2)}}}}$ containing $n+1$ '2s'. (This is a well-known example, mentioned in [27].) For $n \geq 4$ it becomes infeasible to β-normalize such a term (since $2^{2^{2^{2^2}}}$ is 2^{65536}, a number much larger than google). One can express relatively simple theorems using this term such as

$$(\overline{2}^{\alpha^n}\overline{2}^{\alpha^{n-1}}\cdots\overline{2}^{\alpha^0})(\lambda X_\iota X) \overset{*}{=} (\lambda X_\iota X).$$

If one avoids eager β-normalization and allows lemmas, then there is a reasonably short proof using higher-order logic. We first define the set C_2^α of Church numerals (over α) greater than or equal to 2:

$$\lambda N_{(\alpha\to\alpha)\to\alpha\to\alpha}\forall P_{\bullet}(P\overline{2}^\alpha \wedge (\forall M_{\bullet}PM \Rightarrow P(\overline{s}M))) \Rightarrow PN.$$

(Technically, $(\overline{0}\,\overline{2})$ is β-equal to $(\lambda F_{\iota\to\iota}F)$, which is not equal to $\overline{1}$. We work with the set of Church numerals greater than or equal to 2 to avoid this problem.) One can prove two results with little trouble (where the lengths of the proofs do not depend on the type α):

(a) $\forall N_{((\alpha\to\alpha)\to\alpha\to\alpha)\to(\alpha\to\alpha)\to\alpha\to\alpha\bullet}C_2^{\alpha\to\alpha}N \Rightarrow C_2^\alpha(N\overline{2}^\alpha)$
(b) $\forall N_{(\alpha\to\alpha)\to\alpha\to\alpha\bullet}C_2^\alpha N \Rightarrow {}_\bullet(N(\lambda X_\alpha X)) = (\lambda X_\alpha X)$

Using (a) at several types and (b) at type ι, we can prove, e.g.,

$$(\overline{2}^{\alpha^4}\overline{2}^{\alpha^3}\overline{2}^{\alpha^2}\overline{2}^{\alpha^1}\overline{2}^{\alpha^0})(\lambda X_\iota X) \overset{*}{=} (\lambda X_\iota X)$$

in higher-order logic without β-normalizing.

In [13, Chapter 25, p. 376–382] Boolos presents a related example of a first-order problem which has only a very long (practically infeasible) derivation in first-order logic, but which has a short derivation in a second-order logic, by making use of comprehension axioms.

Example 35. (Boolos' Curious Inference)

$$\begin{aligned}
&(\forall n_\bullet f(n,1) = s(1) \wedge \forall x_\bullet f(1,s(x)) = s(s(f(1,x)))\\
&\quad \wedge \forall n_\bullet\forall x_\bullet f(s(n),s(x)) = f(n,f(s(n),x))\\
&\quad \wedge D(1) \wedge \forall x_\bullet(D(x) \Rightarrow D(s(x))))\\
&\Rightarrow D(f(s(s(s(s(1)))),s(s(s(s(1))))))
\end{aligned}$$

If there were an appropriate (first-order) induction principle available, then there should be a short proof of this example. Note that the example specifies f to be the Ackermann function which grows extremely fast and hence $f(s(s(s(s(1)))),s(s(s(s(1)))))$

is a very big number. Actually, there is long first-order proof which is relatively easy to describe. Boolos argues that any first-order proof must be of size at least $2^{(2^{\cdot^{\cdot^{\cdot 2}}})}$ containing 64K '2s' in all (far more enormous than the number 2^{64K} in Example 34). There is no chance of formally representing such a proof with all computation power ever. Boolos presents a short alternative proof in second-order logic that makes use of higher-order lemmas obtained from comprehension axioms. Formulating the appropriate lemmas (as with the lemmas in Example 34) requires human ingenuity that goes beyond the capabilities of what can be supported with primitive substitution and lemma speculation techniques in current theorem proving approaches.

As discussed in [3], there is a family of theorems $\mathbf{A}^1, \mathbf{A}^2, \ldots$ which are all of the same low order such that $\mathbf{A}^n$ is not provable unless one uses set instantiations involving n^{th}-order quantifiers. To obtain concrete examples from the argument, one must use Gödel numbering. A family of simpler examples displaying this phenomenon would likely be enlightening.

7 Conclusion

We have presented a first set of higher-order test examples that may support the development of higher-order proof systems. This set of examples has been structured according to technical aspects and the semantic properties of extensionality and set comprehension. Future work is to add examples and include them in either the TPTP library or an appropriate higher-order variant. Many more examples are particularly needed to illustrate properties of different forms of equality.

References

1. P. B. Andrews. General models and extensionality. *J. of Symbolic Logic*, 37(2):395–397, 1972.
2. P. B. Andrews. On Connections and Higher Order Logic. *J. of Automated Reasoning*, 5:257–291, 1989.
3. P. B. Andrews. Classical type theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 15, pages 965–1007. Elsevier Science, 2001.
4. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Kluwer Academic Publishers, second edition, 2002.
5. P. B. Andrews, M. Bishop, and C. E. Brown. TPS: A theorem proving system for type theory. In D. McAllester, editor, *Proc. of CADE-17*, number 1831 in LNAI, pages 164–169, Pittsburgh, USA, 2000. Springer.
6. Peter B. Andrews. Resolution in type theory. *J. of Symbolic Logic*, 36(3):414–432, 1971.
7. C. Benzmüller. *Equality and Extensionality in Automated Higher-Order Theorem Proving.* PhD thesis, Saarland University, 1999.
8. C. Benzmüller, C. Brown, and M. Kohlhase. Higher-order semantics and extensionality. *J. of Symbolic Logic*, 69(4):1027–1088, 2004.
9. C. Benzmüller, C. E. Brown, and M. Kohlhase. Semantic techniques for higher-order cut-elimination. SEKI Technical Report SR-2004-07, Saarland University, Saarbrücken, Germany, 2004. Available at: `http://www.ags.uni-sb.de/~chris/papers/R37.pdf`.

10. C. Benzmüller and M. Kohlhase. LEO – a higher order theorem prover. In C. Kirchner and H. Kirchner, editors, *Proc. of CADE-15*, number 1421 in LNAI, pages 139–144, Lindau, Germany, 1998. Springer.
11. C. Benzmüller, V. Sorge, M. Jamnik, and M. Kerber. Can a higher-order and a first-order theorem prover cooperate? In F. Baader and A. Voronkov, editors, *Proc. of LPAR 2004*, volume 3452 of *LNAI*, pages 415–431. Springer, 2005.
12. M. P. Bonacina and J. Hsiang. Incompleteness of the RUE/NRF inference systems. Newsletter of the Association for Automated Reasoning, No. 20, pages 9–12, 1992.
13. G. Boolos. *Logic, Logic, Logic*. Harvard University Press, 1998.
14. C. E. Brown. *Set Comprehension in Church's Type Theory*. PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, 2004.
15. A. Church. A formulation of the simple theory of types. *J. of Symbolic Logic*, 5:56–68, 1940.
16. V. J. Digricoli. Resolution by unification and equality. In W. H. Joyner, editor, *Proc. of CADE-4*, Austin, Texas, USA, 1979.
17. Leon Henkin. Completeness in the theory of types. *J. of Symbolic Logic*, 15(2):81–91, 1950.
18. G. P. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
19. M. Kohlhase. *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle*. PhD thesis, Saarland University, 1994.
20. M. Kohlhase. Higher-order tableaux. In *Proc. of TABLEAUX 95*, number 918 in LNAI, pages 294–309. Springer, 1995.
21. J.D. McCharen, R.A. Overbeek, and L.A. Wos. Problems and Experiments for and with Automated Theorem-Proving Programs. *IEEE Transactions on Computers*, C-25(8):773–782, 1976.
22. D. Miller. *Proofs in Higher-Order Logic*. PhD thesis, Carnegie-Mellon Univ., 1983.
23. F.J. Pelletier. Seventy-five Problems for Testing Automatic Theorem Provers. *J. of Automated Reasoning*, 2(2):191–216, 1986.
24. F.J. Pelletier, G. Sutcliffe, and C.B. Suttner. The Development of CASC. *AI Communications*, 15(2-3):79–90, 2002.
25. C. Prehofer. *Solving Higher-Order Equations: From Logic to Programming*. Progress in Theoretical Computer Science. Birkhäuser, 1998.
26. W. Snyder and J. Gallier. Higher-Order Unification Revisited: Complete Sets of Transformations. *J. of Symbolic Computation*, 8:101–140, 1989.
27. R. Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9:73–81, 1979.
28. G. Sutcliffe and C. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *J. of Automated Reasoning*, 21(2):177–203, 1998.
29. G.A. Wilson and J. Minker. Resolution, Refinements, and Search Strategies: A Comparative Study. *IEEE Transactions on Computers*, C-25(8):782–801, 1976.
30. C.-P. Wirth. Descente infinie + Deduction. *Logic J. of the IGPL*, 12(1):1–96, 2004. `www.ags.uni-sb.de/~cp/p/d/welcome.html`.