The Seven Virtues of Simple Type Theory

William M. Farmer*
McMaster University

26 January 2004

Abstract

Simple type theory, also known as *higher-order logic*, is a natural extension of first-order logic which is simple, elegant, highly expressive, and practical. This paper surveys the virtues of simple type theory and attempts to show that simple type theory is an attractive alternative to first-order logic for practical-minded scientists, engineers, and mathematicians. It recommends that simple type theory be incorporated into introductory logic courses offered by mathematics departments.

1 Introduction

Mathematicians are committed to rigorous reasoning, but they usually shy away from formal logic. However, when mathematicians really need a formal logic—e.g., to teach their students the rules of quantification, to pin down exactly what a "property" is, or to formalize set theory—they almost invariably choose some form of *first-order logic*. For mathematicians, as well as scientists and engineers, first-order logic reigns supreme!

A formal logic can be a *theoretical tool* for studying rigorous reasoning and a *practical tool* for performing rigorous reasoning. Today mathematicians sometimes use formal logics as theoretical tools, but they very rarely use them as practical tools. Tomorrow things will be different. In the future mathematicians will routinely employ computer systems—the successors of today's computer algebra systems and computer theorem proving systems—that mechanize various aspects of mathematical reasoning. In order to be

*Address: Department of Computing and Software, McMaster University, 1280 Main Street West, Hamilton, Ontario L8S 4K1, Canada. E-mail: wmfarmer@mcmaster.ca.

trustworthy, these systems will inevitably be based on formal logics. Mathematicians will also utilize huge digital libraries of mathematics. The mathematical knowledge in these libraries will be formalized, organized, certified, searched, and retrieved with the help of formal logics.

First-order logic is certainly an effective logic for *theory*, but in many ways it is an awkward logic for *practice*. Are there any logics that are more effective for practice? A good candidate is an old relative of first-order logic called *simple type theory*.

Simple type theory, also known as *higher-order logic*, is a natural extension of first-order logic. It is based on the same principles as first-order logic but differs from first-order logic in two principal ways. First, terms can be *higher-order*, i.e., they can denote higher-order values such as sets, relations, and functions. Predicates and functions can be applied to higher-order terms, and quantification can be applied to higher-order variables in formulas. Second, syntactic objects called *types*, which denote nonempty sets of values, are used to organize terms. They restrict the scope of variables, control the formation of terms, and provide a means to classify terms by their value.

Simple type theory is a logic with outstanding virtues. It is simple, elegant, highly expressive, and *practical*. Although it is familiar to many computer scientists, most mathematicians, engineers, and other scientists have never heard of it. This is due in large part to the fact that simple type theory is rarely taught in mathematics departments at either the undergraduate or graduate level. However, an understanding of simple type theory would be beneficial to anyone who needs to work with or apply mathematical logic. This is particularly true for:

- Engineers who need to write (and read) precise specifications.
- Computer scientists who employ functional programming languages such as Lisp, ML, and Haskell.
- Software engineers who use higher-order theorem proving systems to model and analyze software systems.
- Mathematics students who are studying the foundations of mathematics or model theory.

In this paper, we will present a pure form of simple type theory that we call STT and then use it to illustrate seven virtues of simple type theory. Our ultimate objective is to show that simple type theory is an attractive alternative to first-order logic for practical-minded scientists, engineers, and

even mathematicians. The paper ends with a recommendation that simple type theory be incorporated into introductory logic courses offered by mathematics departments.

2 History

B. Russell proposed in 1908 [31] a logic now known as the *ramified theory of types*. Russell wanted a logic that would be free from the set-theoretic paradoxes such as Russell's paradox about the class of all classes that are not members of themselves as well as the semantic paradoxes such as Richard's paradox concerning the cardinality of the set of definable real numbers [15]. As a result, the ramified theory of types was formulated with the safety principle (the so-called *vicious-circle principle*) that the class $C = \{x : \varphi(x)\}$ can be defined only if C itself is not in the range of any variable in φ. This safety principle is enforced with a hierarchy of levels of types. Russell and A. Whitehead used the ramified theory of types as the logical basis for their monumental, three-volume *Principia Mathematica* [33], the first attempt to formalize a significant portion of mathematics starting from first principles. The great achievement of *Principia Mathematica*, unfortunately, is marred by the fact that, in order to formalize standard proofs of induction, Russell introduced the Axiom of Reducibility which in effect nullifies the safety principle [12].

In the 1920s, L. Chwistek [5] and F. Ramsey [30] noticed that, if one is willing to give up Russell's safety principle, the hierarchy of levels of types in the ramified theory of types can be collapsed. The resulting simplified logic is called the *simple theory of types* or, more briefly, *simple type theory*. It is equivalent to the ramified theory of types plus the Axiom of Reducibility. Detailed formulations of simple type theory were published in the late 1920s and early 1930s by R. Carnap, K. Gödel, A. Tarski, and W. V. O. Quine (see [13]).

In 1940 [4] A. Church presented an elegant formulation of simple type theory, known as *Church's type theory*, that is based on functions instead of relations and that incorporates special machinery to build and apply functions (lambda-notation and lambda-conversion). Church's paper has had a profound influence on computer science, especially in the areas of programming languages, computer theorem proving, computational logic, and formal methods.[1] Today the field of *type theory* in computer science

[1] In the May 2003 list of the most cited articles in Computer Science on the CiteSeer Web site (`http://citeseer.nj.nec.com/`), Church's paper [4] is ranked 331 with 333

is largely the study of Church-style (classical and constructive) logics that are based on functions and equipped with lambda-notation and lambda-conversion.

Church's type theory has been extensively studied by two of Church's students, L. Henkin and P. Andrews. Henkin proved in [17] that there is a sense in which Church's type theory is complete (see section 7). Henkin also showed in [18] that Church's type theory could be reformulated using only four primitive notions: function application, function abstraction, equality, and definite description (see section 4). Andrews devised in [1] a simple and elegant proof system for Henkin's reformulation of Church's type theory (see section 6). Andrews formulated a version of Church's type theory called $\mathcal{Q}_0$ that employs the ideas developed by Church, Henkin, and himself. Andrews meticulously describes and analyzes $\mathcal{Q}_0$ in his textbook [2], and he and his students have implemented a computer theorem prover based on $\mathcal{Q}_0$ called TPS [3].

Since the 1980s, type theory has been a popular choice for the logical basis of computer theorem proving systems. HOL [14], IMPS [11], Isabelle [27], ProofPower [22], PVS [26], and TPS are examples of systems based on versions of Church's type theory, and Automath [25], Coq [7], LEGO [29], and Nuprl [6] are examples of systems based on constructive type theories.

3 The Definition of STT

There are many variants of simple type theory. STT is a version of Church's type theory [4]. STT is simple type theory boiled down to its essence. It is convenient for study, but it is not highly practical for use. In section 8, we will consider a number of ways that STT can be extended to a more practical form of simple type theory.

We will begin our exploration of simple type theory by defining the syntax and semantics of STT.

3.1 Syntax

STT has two kinds of syntactic objects. "Types" denote nonempty sets of values; they are used to restrict the scope of variables, control the formation of expressions, and classify expressions by value. "Expressions" denote values including the truth values T (true) and F (false); they do what both terms and formulas do in first-order logic.

citations and is the oldest paper with more than 300 citations.

A *type* of STT is defined by the formation rules given below. $\mathbf{type}[\alpha]$ asserts that α is a type.

T1 $\dfrac{}{\mathbf{type}[\iota]}$ **(Type of individuals)**

T2 $\dfrac{}{\mathbf{type}[*]}$ **(Type of truth values)**

T3 $\dfrac{\mathbf{type}[\alpha],\ \mathbf{type}[\beta]}{\mathbf{type}[(\alpha \to \beta)]}$ **(Function type)**

That is, ι and $*$ are atomic types and, whenever α and β are types, $(\alpha \to \beta)$ is a compound type. Let $\mathcal{T}$ denote the set of types of STT.

The *logical symbols* of STT are:

1. *Function application*: @.
2. *Function abstraction*: λ.
3. *Equality*: $=$.
4. *Definite description*: I (capital iota).
5. An infinite set $\mathcal{V}$ of symbols called *variables*.

A *language* of STT is a pair $L = (\mathcal{C}, \tau)$ where:

1. $\mathcal{C}$ is a set of symbols called *constants*.
2. $\mathcal{V}$ and $\mathcal{C}$ are disjoint.
3. $\tau : \mathcal{C} \to \mathcal{T}$ is a total function.

That is, a language is a set of symbols with assigned types (what computer scientists usually call a "signature").

An *expression* E of *type* α of an STT language $L = (\mathcal{C}, \tau)$ is defined by the formation rules given below. $\mathbf{expr}_L[E, \alpha]$ asserts that E is an expression of type α of L.

E1 $\dfrac{x \in \mathcal{V},\ \mathbf{type}[\alpha]}{\mathbf{expr}_L[(x : \alpha), \alpha]}$ **(Variable)**

E2 $\dfrac{c \in \mathcal{C}}{\mathbf{expr}_L[c, \tau(c)]}$ **(Constant)**

E3 $\dfrac{\mathbf{expr}_L[A,\alpha],\ \mathbf{expr}_L[F,(\alpha\to\beta)]}{\mathbf{expr}_L[(F\ @\ A),\beta]}$ **(Function application)**

E4 $\dfrac{x\in\mathcal{V},\ \mathbf{type}[\alpha],\ \mathbf{expr}_L[B,\beta]}{\mathbf{expr}_L[(\lambda\, x:\alpha\ .\ B),(\alpha\to\beta)]}$ **(Function abstraction)**

E5 $\dfrac{\mathbf{expr}_L[E_1,\alpha],\ \mathbf{expr}_L[E_2,\alpha]}{\mathbf{expr}_L[(E_1=E_2),*]}$ **(Equality)**

E6 $\dfrac{x\in\mathcal{V},\ \mathbf{type}[\alpha],\ \mathbf{expr}_L[A,*]}{\mathbf{expr}_L[(\mathrm{I}\, x:\alpha\ .\ A),\alpha]}$ **(Definite description)**

For instance, rule **E1** says that, if x is a variable and α is a type, then $(x:\alpha)$ is an expression of type α. We will see shortly that the value of a definite description $(\mathrm{I}\, x:\alpha\ .\ A)$ is the unique value x of type α satisfying A if it exists and is a canonical "error" value of type α otherwise. "Free variable", "closed expression", and similar notions are defined in the obvious way.

Notice that different kinds of expressions are distinguished *by type* instead of *by form*, as shown by the following examples. An *individual constant* of L is a constant $c\in\mathcal{C}$ such that $\tau(c)=\iota$. A *formula* of L is an expression of L of type $*$. A *predicate* of L is an expression of L of type $(\alpha\to *)$ for any $\alpha\in\mathcal{T}$.

Let $A_\alpha, B_\alpha, C_\alpha, \ldots$ denote expressions of type α. Parentheses and the types of variables may be dropped when meaning is not lost. An expression of the form $(F\ @\ A)$ will usually be written in the more compact and standard form $F(A)$.

Virtue 1 STT *has a simple and highly uniform syntax.*

One can argue that the syntax of STT is simpler and more uniform than the syntax of first-order logic with its variables, individual constants, function and predicate symbols, propositional connectives, and quantifiers.

3.2 Semantics

The semantics of STT is based on "standard models". Later in the paper we will introduce another semantics based on "general models".

A *standard model* for a language $L=(\mathcal{C},\tau)$ of STT is a triple $M=(\mathcal{D},\mathcal{E},I)$ where:

1. $\mathcal{D}=\{D_\alpha:\alpha\in\mathcal{T}\}$ is a set of nonempty domains (sets).

2. $D_* = \{\mathrm{T}, \mathrm{F}\}$, the domain of truth values.

3. For $\alpha, \beta \in \mathcal{T}$, $D_{\alpha\to\beta}$ is the set of *all* functions from D_α to D_β.

4. $\mathcal{E} = \{e_\alpha : \alpha \in \mathcal{T}\}$ is a set of values such that $e_\alpha \in D_\alpha$ for each $\alpha \in \mathcal{T}$.

5. I maps each $c \in \mathcal{C}$ to a member of $D_{\tau(c)}$.

For each $\alpha \in \mathcal{T}$, e_α is intended to be a canonical "error" value of type α. We say that M is *infinite* if D_ι is infinite.

Fix a standard model $M = (\mathcal{D}, \mathcal{E}, I)$ for a language $L = (\mathcal{C}, \tau)$ of STT. A *variable assignment* into M is a function that maps each variable expression $(x : \alpha)$ to a member of D_α. Given a variable assignment φ into M, an expression $(x : \alpha)$, and $d \in D_\alpha$, let $\varphi[(x : \alpha) \mapsto d]$ be the variable assignment φ' into M such that $\varphi'((x : \alpha)) = d$ and $\varphi'(v) = \varphi(v)$ for all $v \neq (x : \alpha)$.

The *valuation function* for M is the binary function V^M that satisfies the following conditions for all variable assignments φ into M and all expressions E of L:

1. Let E be of the form $(x : \alpha)$. Then $V^M_\varphi(E) = \varphi((x : \alpha))$.

2. Let $E \in \mathcal{C}$. Then $V^M_\varphi(E) = I(E)$.

3. Let E be of the form $(F\ @\ A)$. Then $V^M_\varphi(E) = V^M_\varphi(F)(V^M_\varphi(A))$, the result of applying the function $V^M_\varphi(F)$ to the argument $V^M_\varphi(A)$.

4. Let E be of the form $(\lambda\, x : \alpha\ .\ B)$ where B is of type β. Then $V^M_\varphi(E)$ is the function $f : D_\alpha \to D_\beta$ such that, for each $d \in D_\alpha$, $f(d) = V^M_{\varphi[(x:\alpha)\mapsto d]}(B)$.[2]

5. Let E be of the form $(E_1 = E_2)$. If $V^M_\varphi(E_1) = V^M_\varphi(E_2)$, then $V^M_\varphi(E) = \mathrm{T}$; otherwise $V^M_\varphi(E) = \mathrm{F}$.

6. Let E be of the form $(\mathrm{I}\, x : \alpha\ .\ A)$. If there is a unique $d \in D_\alpha$ such that $V^M_{\varphi[(x:\alpha)\mapsto d]}(A) = \mathrm{T}$, then $V^M_\varphi(E) = d$; otherwise $V^M_\varphi(E) = e_\alpha$.[2]

Let E be an expression of type α of L and A be a formula of L. $V^M_\varphi(E)$ is called the *value* of E in M with respect to φ. $V^M_\varphi(E) \in D_\alpha$, and if E is closed, $V^M_\varphi(E)$ does not depend on φ. A value $d \in D_\alpha$ is *nameable* if there is

[2]Notice that the semantics for function abstraction and definite description is defined using the same trick (due to Tarski) that is used to define the semantics for universal and existential quantification in first-order logic.

some closed expression E of L such that d is the value of E in M. A is *valid* in M, written $M \models A$, if $V_\varphi^M(A) = \mathrm{T}$ for all variable assignments φ into M. A *sentence* of L is a closed formula of L. A is a *semantic consequence* of a set Σ of sentences of L, written $\Sigma \models A$, if $M \models A$ for every standard model M for L such that $M \models B$ for all $B \in \Sigma$.

A *theory* of STT is a pair $T = (L, \Gamma)$ where L is a language of STT and Γ is a set of sentences of L called the *axioms* of T. A formula A is a *semantic consequence* of T, written $T \models A$, if $\Gamma \models A$. A *standard model* of T is a standard model M for L such that $M \models B$ for all $B \in \Gamma$.

Virtue 2 *The semantics of* STT *is based on a small collection of well-established ideas.*

The two semantics of first-order logic and STT are based on essentially the same ideas: domains of individuals, truth values, and functions; models for languages; variable assignments; and valuation functions defined recursively on the syntax of expressions.

4 Expressivity

On the surface, the expressivity of STT appears to be rather modest. The formulas of STT are limited to variables, constants, and definite descriptions of type $*$, equations between expressions of the same type, and applications of predicates; there are no propositional connectives nor quantifiers. However, STT can be used to reason about an infinite hierarchy of higher-order functions constructed over a domain of individuals and a domain of truth values. Since sets can be represented by predicates, the hierarchy includes a subhierarchy of higher-order sets constructed from individuals and truth values. Hidden within this hierarchy is the full power of higher-order predicate logic.

As Henkin shows in [18], the usual propositional connectives and quantifiers can be defined in a logic like STT using just function application, function abstraction, and equality. Here are their definitions in STT:

T	means	$(\lambda\, x : * \,.\, x) = (\lambda\, x : * \,.\, x)$
F	means	$(\lambda\, x : * \,.\, \mathsf{T}) = (\lambda\, x : * \,.\, x)$
$(\neg A_*)$	means	$A_* = \mathsf{F}$
$(A_\alpha \neq B_\alpha)$	means	$\neg(A_\alpha = B_\alpha)$
$(A_* \wedge B_*)$	means	$(\lambda\, f : (* \to (* \to *)) \,.\, f(\mathsf{T})(\mathsf{T})) =$
		$(\lambda\, f : (* \to (* \to *)) \,.\, f(A_*)(B_*))$

$(A_* \vee B_*)$	means	$\neg(\neg A_* \wedge \neg B_*)$
$(A_* \Rightarrow B_*)$	means	$\neg A_* \vee B_*$
$(A_* \Leftrightarrow B_*)$	means	$A_* = B_*$
$\Pi_{(\alpha\to*)\to*}$	means	$\lambda\, p : (\alpha \to *)\ .\ p = (\lambda\, x : \alpha\ .\ \mathsf{T})$
$(\forall\, x : \alpha\ .\ A_*)$	means	$\Pi_{(\alpha\to*)\to*}(\lambda\, x : \alpha\ .\ A_*)$
$(\exists\, x : \alpha\ .\ A_*)$	means	$\neg(\forall\, x : \alpha\ .\ \neg A_*)$
$\bot_\alpha$	means	$\mathrm{I}\, x : \alpha\ .\ x \neq x$
$\mathsf{if}(A_*, B_\alpha, C_\alpha)$	means	$\mathrm{I}\, x : \alpha\ .\ (A_* \Rightarrow x = B_\alpha) \wedge (\neg A_* \Rightarrow x = C_\alpha)$ where x does not occur in A_*, B_α, or C_α

Notice that we are using the syntactic conventions mentioned in section 3.1. For example, the meaning of T is officially the expression

$$((\lambda\, x : *\ .\ (x : *)) = (\lambda\, x : *\ .\ (x : *))).$$

In addition to the definitions of the usual propositional connectives and quantifiers, we also included above two definitions that employ definition description. $\bot_\alpha$ is a canonical error expression of type α. if is an if-then-else expression constructor (i.e., $\mathsf{if}(A_*, B_\alpha, C_\alpha)$ denotes B_α if A_* holds and denotes C_α if A_* does not hold).

If we fix the type of a variable x, say to α, then an expression of the form $\Box\, x : \alpha\ .\ E$ may be written as simply $\Box\, x\ .\ E$ where $\Box$ is λ, I, $\forall$, or $\exists$. If desired, all types can be removed from an expression by fixing the types of the variables occurring in the expression. We will write a formula of the form $\Box\, x_1 : \alpha\ .\ \cdots\ \Box\, x_n : \alpha\ .\ A$ as simply $\Box\, x_1, \ldots, x_n : \alpha\ .\ A$ where $\Box$ is $\forall$ or $\exists$. Similarly, we will write a formula of the form $\Box\, x_1\ .\ \cdots\ \Box\, x_n\ .\ A$ (where the types of $x_1, \ldots, x_n$ have been fixed) as simply $\Box\, x_1, \ldots, x_n\ .\ A$ where $\Box$ is $\forall$ or $\exists$.

Using these definitions, first-order logic, second-order logic, third-order logic, etc. can be "embedded" in STT. The precise statement of this result is:

Theorem 1 *Let T be any theory of nth-order logic for any $n \geq 1$. Then there is a theory T' of* STT *such that there is a faithful interpretation of T in T' (i.e., there is a translation Φ from the sentences of T to the sentences of T' such that, for all sentences A of T, $T \models A$ iff $T' \models \Phi(A)$).*

Because STT is equipped with full higher-order quantification and definite description, many mathematical notions can be directly and naturally expressed in STT. We will give five simple examples, three in this section and two in the next section, that illustrate how "higher-order" concepts can

be expressed in STT. None of these examples can be directly expressed in first-order logic.

For the first example, let equiv-rel be the expression

$$\begin{array}{l}\lambda\, p : (\iota \to (\iota \to *)) \;. \\ \quad \forall\, x : \iota \;.\; p(x)(x) \wedge \\ \quad \forall\, x, y : \iota \;.\; p(x)(y) \Rightarrow p(y)(x) \wedge \\ \quad \forall\, x, y, z : \iota \;.\; (p(x)(y) \wedge p(y)(z)) \Rightarrow p(x)(z).\end{array}$$

Then equiv-rel(r) asserts that a binary relation r represented (in curryed form) as a function of type $(\iota \to (\iota \to *))$ is an equivalence relation.

For the second example, let compose be the expression

$$\lambda\, f : (\iota \to \iota) \;.\; \lambda\, g : (\iota \to \iota) \;.\; \lambda\, x : \iota \;.\; f(g(x)).$$

If f, g are expressions of type $(\iota \to \iota)$, then compose$(f)(g)$ is an expression that denotes the composition of f and g as functions.

For the third example, let inv-image be the expression

$$\lambda\, f : (\iota \to \iota) \;.\; \lambda\, s : (\iota \to *) \;.\; \mathrm{I}\, s' : (\iota \to *) \;.\; \forall\, x : \iota \;.\; s'(x) \Leftrightarrow s(f(x)).$$

If f is an expression of type $(\iota \to \iota)$ representing a function and s is an expression of type $(\iota \to *)$ representing a set, then inv-image$(f)(s)$ is an expression of type $(\iota \to *)$ that represents the inverse image of s under f.

Virtue 3 STT *is a highly expressive logic.*

In fact, nearly all theorems of mathematics can be straightforwardly expressed in STT.

5 Categoricity

Theories are used to specify structures. Some theories (e.g., a theory of groups) specify collections of structures (e.g., the collection of all groups). Other theories (e.g., the theory of a complete ordered field) specify a single structure (e.g., the ordered field of the real numbers). In other words, a theory specifies the set of models that belong to the theory. A theory is *categorical* if it has exactly one model up to isomorphism. Categorical theories are very desirable in applications in which a theory is used as a "model" of a specific complex system.

In his 1889 booklet [28], G. Peano presented a theory of natural number arithmetic, commonly called *Peano Arithmetic*. The theory consists of

Peano's famous five axioms for the natural numbers plus four axioms for equality. Independently of Peano, R. Dedekind developed in [8] a theory of natural number arithmetic very similar to Peano Arithmetic. He proved, in effect, that all structures satisfying Peano's axioms are isomorphic to the standard structure $(\mathbf{N}, 0, s)$ of the natural numbers, i.e., that Peano Arithmetic is categorical.

Let $\mathsf{PA} = (L, \Gamma)$ be the theory of STT where $L = (\{0, s\}, \tau)$, $\Gamma = \{A_1, A_2, A_3\}$,

1. $\tau(0) = \iota$,
2. $\tau(s) = \iota \to \iota$,
3. A_1 is $\quad \forall\, x : \iota \,.\, s(x) \neq 0$,
4. A_2 is $\quad \forall\, x, y : \iota \,.\, s(x) = s(y) \Rightarrow x = y$, $\quad$ and
5. A_3 is $\quad \forall\, p : (\iota \to *) \,.\, (p(0) \wedge (\forall\, x : \iota \,.\, p(x) \Rightarrow p(s(x)))) \Rightarrow \forall\, x : \iota \,.\, p(x)$.

PA is a very direct formalization of Peano Arithmetic. The type ι denotes the set $\mathbf{N} = \{0, 1, 2, \ldots\}$ of the natural numbers, 0 denotes the first natural number, and s denotes the successor function. The five clauses of the definition of PA correspond to Peano's five axioms for the natural numbers. (A_1 says 0 is not a successor, A_2 says s is injective, and A_3 expresses the induction principle.) The other basic operations of natural number arithmetic, such as $+$, $\cdot$, and $<$, can be defined in PA.

Like Peano Arithmetic itself, PA is categorical. Its unique standard model (up to isomorphism) is $(\mathcal{D}, \mathcal{E}, I)$ where $D_\iota = \{0, 1, 2, \ldots\}$, $\mathcal{E}$ is any set of canonical error values, $I(0) = 0$, and $I(s) =$ the successor function on D_ι.

Peano Arithmetic cannot be directly formalized in first-order logic because the induction principle involves quantification over predicates, which is not directly expressible in first-order logic. There is, however, a standard first-order formalization $\mathsf{PA}' = (L', \Gamma')$ of Peano Arithmetic where L' is the first-order language containing a constant 0, a unary function symbol s, and two binary function symbols $+$ and $\cdot$ and where Γ is the following set of formulas of L':

1. $\forall\, x \,.\, s(x) \neq 0$.
2. $\forall\, x, y \,.\, s(x) = s(y) \Rightarrow x = y$.
3. $\forall\, x \,.\, x + 0 = x$.

4. $\forall\, x, y\ .\ x + s(y) = s(x + y)$.

5. $\forall\, x\ .\ x \cdot 0 = 0$.

6. $\forall\, x, y\ .\ x \cdot s(y) = (x \cdot y) + x$.

7. $(A[0] \wedge (\forall\, x\ .\ A[x] \Rightarrow A[s(x)])) \Rightarrow \forall\, x\ .\ A[x]$
   for each formula $A[x]$ of L'.

Clause 7 is an infinite collection of formulas called the *induction schema*. It is only an approximation of the induction principle. In fact, the induction schema includes just a countably infinite number of instances of the induction principle, while the induction principle has a continuum number of instances, one for each property of the natural numbers.

PA' is not categorical. Since PA' does not contain all instances of the induction principle, Dedekind's proof of the categoricity of Peano Arithmetic fails. Moreover, PA' has "nonstandard" models containing infinite natural numbers by the compactness theorem of first-order logic.

The failure to achieve categoricity for a theory of natural number arithmetic in first-order logic is not an aberration. Rather it is an instance of a fundamental weakness of first-order logic that is a simple consequence of the compactness theorem of first-order logic:

Theorem 2 *Any first-order theory that has an infinite model has infinitely many (infinite) nonisomorphic models.*

Thus, a first-order theory that is intended to specify a single infinite structure cannot be categorical.

Let us consider a second example. It is well known that there is exactly one *complete ordered field* up to isomorphism, namely, the standard structure

$$(\mathbf{R}, +, 0, -, \cdot, 1, ^{-1}, \mathsf{pos})$$

of the real numbers where pos denotes the set of positive real numbers.

Let $\mathsf{COF} = (L, \Gamma)$ be the theory of STT such that:

- $L = (\{+, 0, -, \cdot, 1, ^{-1}, \mathsf{pos}, <, \leq, \mathsf{ub}, \mathsf{lub}\}, \tau)$ where τ is defined by:

Constant c	**Type** $\tau(c)$
0	ι
1	ι
$-$	$\iota \to \iota$
$^{-1}$	$\iota \to \iota$
pos	$\iota \to *$
$+$	$\iota \to (\iota \to \iota)$
$\cdot$	$\iota \to (\iota \to \iota)$
$<$	$\iota \to (\iota \to *)$
$\leq$	$\iota \to (\iota \to *)$
ub	$(\iota \to *) \to (\iota \to *)$
lub	$(\iota \to *) \to (\iota \to *)$

- Γ is the set of the 18 formulas given below. We assume that the variables x, y, z are of type ι and the variable s is of type $(\iota \to *)$.

1. $\forall\, x, y, z\ .\ (x + y) + z = x + (y + z)$.
2. $\forall\, x, y\ .\ x + y = y + x$.
3. $\forall\, x\ .\ x + 0 = x$.
4. $\forall\, x\ .\ x + (-x) = 0$.
5. $\forall\, x, y, z\ .\ (x \cdot y) \cdot z = x \cdot (y \cdot z)$.
6. $\forall\, x, y\ .\ x \cdot y = y \cdot x$.
7. $\forall\, x\ .\ x \cdot 1 = x$.
8. $\forall\, x\ .\ x \neq 0 \Rightarrow x \cdot x^{-1} = 1$.
9. $0 \neq 1$.
10. $\forall\, x, y, z\ .\ x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
11. $\forall\, x\ .\ (x = 0 \wedge \neg\mathsf{pos}(x) \wedge \neg\mathsf{pos}(-x)) \vee$
    $(x \neq 0 \wedge \mathsf{pos}(x) \wedge \neg\mathsf{pos}(-x)) \vee$
    $(x \neq 0 \wedge \neg\mathsf{pos}(x) \wedge \mathsf{pos}(-x))$.
12. $\forall\, x, y\ .\ (\mathsf{pos}(x) \wedge \mathsf{pos}(y)) \Rightarrow \mathsf{pos}(x + y)$.
13. $\forall\, x, y\ .\ (\mathsf{pos}(x) \wedge \mathsf{pos}(y)) \Rightarrow \mathsf{pos}(x \cdot y)$.
14. $\forall\, x, y\ .\ x < y \Leftrightarrow \mathsf{pos}(y - x)$.
15. $\forall\, x, y\ .\ x \leq y \Leftrightarrow (x < y \vee x = y)$.
16. $\forall\, s, x\ .\ x\ \mathsf{ub}\ s \Leftrightarrow (\forall\, y\ .\ s(y) \Rightarrow y \leq x)$.
17. $\forall\, s, x\ .\ x\ \mathsf{lub}\ s \Leftrightarrow (x\ \mathsf{ub}\ s \wedge (\forall\, y\ .\ y\ \mathsf{ub}\ s \Rightarrow x \leq y))$.
18. $\forall\, s\ .\ ((\exists\, x\ .\ s(x)) \wedge (\exists\, x\ .\ x\ \mathsf{ub}\ s)) \Rightarrow \exists\, x\ .\ x\ \mathsf{lub}\ s$.

Notes:

1. As a result of fixing the types for the variables x, y, z, s, the axioms of COF are free of types and thus look just like the axioms one might see in any mathematics textbook.

2. We write the additive and multiplicative inverses of x as $-x$ and x^{-1} instead of as $-(x)$ and $^{-1}(x)$, respectively.

3. $+$ and $*$ are formalized by constants of type $(\iota \to (\iota \to \iota))$ representing curryed functions. However, we write the application of $+$ and $*$ using infix notation, e.g., we write $x + y$ instead of $+(x)(y)$. $<$ and $\leq$ are handled in a similar way.

4. pos is a predicate that represents the set of positive real numbers.

5. $\mathsf{ub}(s)(x)$ and $\mathsf{lub}(s)(x)$ say that x is an upper bound of s and x is the least upper bound of s, respectively. We write $\mathsf{ub}(s)(x)$ and $\mathsf{lub}(s)(x)$ as x ub s and x lub s, respectively, using infix notation.

6. Formula 18 expresses the *completeness principle* of the real numbers, i.e., that every set of real numbers that is nonempty and has an upper bound has a least upper bound.

COF is a direct formalization of the theory of a complete ordered field. COF is categorical; its unique standard model (up to isomorphism) is $(\mathcal{D}, \mathcal{E}, I)$ where $D_\iota = \mathbf{R}$, the set of real numbers, $\mathcal{E}$ is any set of canonical error values, and I assigns $+$, 0, $-$, $\cdot$, 1, $^{-1}$, pos, $<$, $\leq$, ub, lub their usual meanings.

Like Peano Arithmetic, the theory of a complete ordered field is a theory of a single fundamental infinite structure that cannot be directly formalized in first-order logic as a categorical theory since the completeness principle involves quantification over predicates.

Virtue 4 STT *admits categorical theories of infinite structures.*

It is worthwhile to note that PA and COF are both formalized using only a small portion of the machinery of STT. In fact, both Peano Arithmetic and the theory of a complete ordered field can be formalized in *second-order logic* as categorical theories.

We will conclude this section by giving the fourth and fifth examples that illustrate the expressivity of STT.

For the fourth example, let abs be the expression

$$\lambda r : \iota \,.\, \mathsf{if}(0 \leq r, r, -r)$$

in COF. abs denotes the absolute value function on the real numbers.

For the fifth example, let lim be the expression

$$\begin{array}{l}\lambda\, f : (\iota \to \iota)\ .\ \lambda\, a : \iota\ . \\ \quad (\mathrm{I}\, l : \iota\ .\ (\forall\, \epsilon : \iota\ .\ 0 < \epsilon \Rightarrow \\ \quad\quad (\exists\, \delta : \iota\ .\ 0 < \delta \wedge \\ \quad\quad\quad (\forall\, x : \iota\ .\ (\mathsf{abs}(x - a) < \delta \wedge x \neq a) \Rightarrow \mathsf{abs}(f(x) - l) < \epsilon))))\end{array}$$

in COF. Suppose f is an expression of type of $(\iota \to \iota)$ and a is an expression of type ι in COF. Then $\mathsf{lim}(f)(a)$ denotes the limit of f at a if f approaches a limit at a; otherwise $\mathsf{lim}(f)(a)$ denotes the canonical error value of type ι.

6 Provability

Up to this point we have said nothing about formal proof. In this section we investigate provability in STT. We begin by giving some definitions.

A *(Hilbert-style) proof system* for STT consists of a finite set of axiom schemas and rules of inference. Let $\mathbf{P}$ be a proof system for STT and $T = (L, \Gamma)$ be a theory of STT. A finite sequence of formulas of L is a *proof* of a formula A from T in $\mathbf{P}$ if A is the last member of the sequence and every member of the sequence is an instance of one of the axiom schemas of $\mathbf{P}$, a member of Γ, or is inferred from previous members by a rule of inference of $\mathbf{P}$.

Let $T \vdash_{\mathbf{P}} A$ mean there is a proof of A from T in $\mathbf{P}$. $\mathbf{P}$ is *sound* if, for every theory $T = (L, \Gamma)$ and formula A of L,

$$T \vdash_{\mathbf{P}} A \ \text{ implies } \ T \models A.$$

$\mathbf{P}$ is *complete* if, for every theory $T = (L, \Gamma)$ and formula A of L,

$$T \models A \ \text{ implies } \ T \vdash_{\mathbf{P}} A.$$

It is well known that there is no sound and complete proof system for second-order logic, and hence, it is not surprising that there is no sound and complete proof system for STT. On the surface, this may appear to be a weakness of STT, but indeed it is a sign of its strength. The incompleteness of STT is an immediate consequence of Gödel's Incompleteness Theorem.

Theorem 3 (Incompleteness) *There is no sound and complete proof system for* STT.

Proof Suppose **P** is a sound and complete proof system for STT. By the soundness of **P** and Gödel's Incompleteness Theorem, there is a sentence A such that (1) $M \models A$, where M is the unique standard model for PA (up to isomorphism), and (2) $\mathsf{PA} \not\vdash_{\mathbf{P}} A$. By the completeness of **P**, (2) implies $\mathsf{PA} \not\models A$ and hence $M \not\models A$ since M is the only standard model of PA, which contradicts (1). □

We will now present a very simple and elegant proof system for L called **A** which is adapted for STT from a proof system devised by Andrews [1]. Define $B_\beta[(x:\alpha) \mapsto A_\alpha]$ to be the result of simultaneously replacing each free occurrence of $(x:\alpha)$ in B_β by an occurrence of A_α. Let $(\exists\,!\,x:\alpha\,.\,A)$ mean

$$((\exists\,x:\alpha\,.\,A) \wedge (\forall\,y:\alpha\,.\,A[(x:\alpha) \mapsto (y:\alpha)] \Rightarrow y = x))$$

where y does not occur in A. This formula asserts there exists a unique value x of type α that satisfies A.

A consists of the following six axiom schemas and single rule of inference:

A1 (Truth Values)

$$\forall\,f:(*\to *)\,.\,(f(\mathsf{T}) \wedge f(\mathsf{F})) \Leftrightarrow (\forall\,x:*\,.\,f(x)).$$

A2 (Leibniz' Law)

$$\forall\,x,y:\alpha\,.\,(x=y) \Rightarrow (\forall\,p:(\alpha\to *)\,.\,p(x) \Leftrightarrow p(y)).$$

A3 (Extensionality)

$$\forall\,f,g:(\alpha\to\beta)\,.\,(f=g) \Leftrightarrow (\forall\,x:\beta\,.\,f(x)=g(x)).$$

A4 (Beta-Reduction)

$$(\lambda\,x:\alpha\,.\,B_\beta)(A_\alpha) = B_\beta[(x:\alpha)\mapsto A_\alpha]$$

provided A_α is free for $(x:\alpha)$ in B_β.

A5 (Proper Definite Description)

$$(\exists\,!\,x:\alpha\,.\,A_*) \Rightarrow A_*[(x:\alpha)\mapsto(\mathrm{I}\,x:\alpha\,.\,A_*)]$$

provided $(\mathrm{I}\,x:\alpha\,.\,A_*)$ is free for $(x:\alpha)$ in A_*.

A6 (Improper Definite Description)

$$\neg(\exists\,!\,x : \alpha\,.\,A_*) \Rightarrow (\mathrm{I}\,x : \alpha\,.\,A_*) = \bot_\alpha.$$

R (Equality Substitution) From $A_\alpha = B_\alpha$ and C_* infer the result of replacing one occurrence of A_α in C_* by an occurrence of B_α, provided that the occurrence of A_α in C_* is not immediately preceded by λ.

Notice that **A** does not include a comprehension axiom schema for defining functions from expressions. It is unnecessary because a function can be defined directly from an expression via function abstraction. Moreover, the comprehensive axiom schema is provable in **A** (see the proof of Theorem 5243 in [2] for details).

Theorem 4 **A** *is sound.*

Proof Each instance of the axiom schemas **A1**, **A2**, **A3**, **A4**, **A5**, **A6** is valid in every standard model, and rule **R** preserves validity in every standard model. For details, see the proof of Theorem 5402 in [2]. □

Theorem 5 **A** *is not complete.*

Proof Corollary of Theorem 3. □

Since **A** is incomplete, it is not obvious whether the basic theorems of simple type theory can be proven in **A**. Does **A** have sufficient provability power to be useful? The answer is yes indeed—**A** has enough provability power to serve as a foundation for mathematics.

Let **A+I** be **A** plus an additional axiom that says that the domain of individuals is infinite.

Theorem 6 **A+I** *is equiconsistent with bounded Zermelo set theory.*

Bounded Zermelo set theory is Zermelo set theory (i.e., Zermelo-Fraenkel set theory without the replacement axiom) with a version of the separation axiom in which the quantifiers are bounded by sets. S. Mac Lane has advocated bounded Zermelo set theory as an adequate foundation for mathematics [23], and as a result, bounded Zermelo set theory is commonly known as *Mac Lane set theory.*

A variant of Theorem 6 was first proved by R. Jensen [20]; a more detailed proof is found in A. Mathias's paper [24].

Virtue 5 *There is a proof system for* STT *that is simple, elegant, and powerful.*

In the next section, we will show that there is a sense in which **A** is actually complete—which should expel any queasiness that one has about how well **A** captures the semantics of STT.

7 General Models

When Henkin was a graduate student at Princeton, he investigated the structure of nameable values (see section 3.2) in standard models of Church's type theory [19]. His investigation lead to two extraordinary discoveries.

First, he discovered that the proof system for Church's type theory is actually sound and complete if the semantics is generalized by substituting the notion of a "general model" for the notion of a standard model [17]. Recall that the incompleteness of Church's type theory with respect to the ordinary semantics based on standard models follows from Gödel's incompleteness theorem (see Theorem 3).

Second, he found that the method employed in his proof of the generalized completeness of Church's type theory could be used to get a new proof of the completeness of first-order logic (which was first proved by Gödel) [16]. This proof is now one of the hallmarks of first-order model theory.

A *general structure* for a language $L = (\mathcal{C}, \tau)$ of STT is a triple $M = (\mathcal{D}, \mathcal{E}, I)$ where:

1. $\mathcal{D} = \{D_\alpha : \alpha \in \mathcal{T}\}$ is a set of nonempty domains (sets).
2. $D_* = \{\mathrm{T}, \mathrm{F}\}$.
3. For $\alpha, \beta \in \mathcal{T}$, $D_{\alpha\to\beta}$ is some nonempty set of functions from D_α to D_β.
4. $\mathcal{E} = \{e_\alpha : \alpha \in \mathcal{T}\}$ is a set of values such that $e_\alpha \in D_\alpha$ for each $\alpha \in \mathcal{T}$.
5. I maps each $c \in \mathcal{C}$ to a member of $D_{\tau(c)}$.

M is a *general model* for L if there is a binary function V^M that satisfies the same conditions as the valuation function for a standard model. A general model is thus the same as a standard model except that the function domains of the model may not be "fully inhabited". Hence every standard model for L is also a general model for L. Let us say that M is a *nonstandard model* for L if it is a general model, but not a standard model, for L.

Let $T = (L, \Gamma)$ be a theory of STT and A be a formula of L. A is a *semantic consequence* of a set Σ of sentences of L *in the general sense*, written $\Sigma \models_g A$, if $M \models A$ for every general model M for L such that $M \models B$ for all $B \in \Sigma$. A is a *semantic consequence* of T *in the general sense*, written $T \models_g A$, if $\Gamma \models_g A$. A *general model* of T is a general model M for L such that $M \models B$ for all $B \in \Gamma$.

Let $\mathbf{P}$ be a proof system for STT. $\mathbf{P}$ is *sound in the general sense* if, for every theory $T = (L, \Gamma)$ and formula A of L,

$$T \vdash_{\mathbf{P}} A \text{ implies } T \models_g A.$$

$\mathbf{P}$ is *complete in the general sense* if, for every theory $T = (L, \Gamma)$ and formula A of L,

$$T \models_g A \text{ implies } T \vdash_{\mathbf{P}} A.$$

Theorem 7 $\mathbf{A}$ *is sound in the general sense.*

Proof This is a generalization of Theorem 4; see the proof of Theorem 5402 in [2]. □

A theory T is *consistent* if there is no proof of F from T in $\mathbf{A}$. For a language $L = (\mathcal{C}, \tau)$, the cardinality of L, written $|L|$, is $|\mathcal{C}| + \aleph_0$. A general model $(\{D_\alpha : \alpha \in \mathcal{T}\}, \mathcal{E}, I)$ for L is *frugal* if $|D_\alpha| \leq |L|$ for all $\alpha \in \mathcal{T}$.

Theorem 8 (Henkin's Theorem) *Every consistent theory of* STT *has a frugal general model.*

Proof See the proof of Theorem 5501 in [2]. □

Corollary 1 *There is a general model of* $\mathsf{COF} = (L, \Gamma)$ *such that* $|D_\iota| = \aleph_0$ *(provided* COF *is consistent).*

Thus by Henkin's Theorem, there exist nonstandard models of COF, the theory of the real numbers, in which D_ι contains only a countable number of real numbers. Such models result because $D_{\iota \to *}$ does not contain every predicate, and as a result, the completeness principle says that only the nonempty, bounded subsets of D_ι corresponding to members of $D_{\iota \to *}$ have least upper bounds.

Theorem 9 (Henkin's Completeness Theorem) $\mathbf{A}$ *is complete in the general sense.*

Proof Follows from Henkin's Theorem by an easy argument; see [2]. □

Analogs of the basic theorems of first-order model theory—such as the compactness theorem and the Löwenheim-Skolem theorem—can be derived from Theorems 8 and 9.

Nonstandard models in STT have the same importance to the model theory of STT as nonstandard first-order models have to first-order model theory. Each nonstandard model of a theory exhibits a way of interpreting what the theory means. Some nonstandard models (like the countable model of COF mentioned above) expose semantic defects (of the theory or model). Others (like a model of COF which contains infinitesimals) provide foundations for different ways of analyzing the standard model(s) of the theory (as is done in nonstandard analysis).

In STT, the distinction between standard and nonstandard models is absolutely clear: in a standard model every function domain is fully inhabited, while in a nonstandard model some function domain is only partially inhabited. However, in first-order logic, there is no formal distinction between the standard and nonstandard models of a theory. In fact, for the first-order theories of the natural numbers and the real numbers, this distinction is meaningless without a "higher-order" perspective.

For example, consider a general model M of PA that includes nonstandard (infinite) natural numbers. We know such models exist by the generalized compactness theorem of STT. Is M standard or nonstandard? Since M is a general model of PA, the induction principle A, which has the form $(\forall\, p : (\iota \to *)\ .\ B)$, is valid in M. If d is the predicate that is true for all the standard (finite) natural numbers but false for all nonstandard natural numbers, then $V^M_{\varphi[(p:\iota\to *)\mapsto d]}(B) = \mathrm{F}$ for any variable assignment φ into M, which contradicts the validity of A in M. Therefore, $d \notin D_{\iota\to *}$, and thus M is nonstandard.

Now consider a model of the first-order theory PA′ that includes nonstandard natural numbers. We know such models exist by the compactness theorem of first-order logic. Is M standard or nonstandard? The theory PA′ alone does not provide us with any way of distinguishing standard models from nonstandard models. The distinction actually relies on the following two facts:

1. PA′ is intended to be a first-order approximation of Peano Arithmetic.

2. The usual model of the natural numbers is the only model of Peano Arithmetic (up to isomorphism).

Therefore, M is a nonstandard model of PA' since it is not the unique model of Peano Arithmetic (up to isomorphism). Notice that to distinguish between standard and nonstandard models of PA' we had to appeal to Peano Arithmetic, a "higher-order" theory.

Virtue 6 *Henkin's general models semantics enables the techniques of first-order model theory to be applied to* STT *and illuminates the distinction between standard and nonstandard models.*

8 Practicality

Is STT a practical logic for formalizing ordinary mathematics? The answer is no. Although mathematical ideas can be formalized much more directly and succinctly in STT than in first-order logic, formalizing real mathematics in STT would be quite burdensome. However, by extending the syntax and semantics of STT in certain ways, STT can be made into an effective logic for actual use. These extensions do not change STT in any fundamental way; they just make STT more convenient to employ.

Many practical ways of extending a simple type theory like STT have been proposed, beginning with [32]. Here is a list of examples:

1. STT can be made into a "many-sorted" logic by allowing a language to include a set of *base types*, each of which denotes a type of individuals.

2. Machinery for working with basic mathematical objects like sets, tuples, and lists can be built into STT by adding new type constructors and new expression constructors or built-in constants. For example, see the logic BESTT [10].

3. An operator is *polymorphic* if it can be applied to expressions of more than one type (e.g., $=$). Polymorphic operators can be admitted to STT by introducing type variables. For example, a constant $\mathsf{compose}$ that gives the composition of two functions of appropriate type could have the type

$$((\nu \to \xi) \to ((\mu \to \nu) \to (\mu \to \xi)))$$

where μ, ν, ξ are type variables. Given an expression f of type $(\beta \to \gamma)$ and another expression g of type $(\alpha \to \beta)$, then $\mathsf{compose}(f)(g)$ would return an expression of type $(\alpha \to \gamma)$. See the logic of the HOL theorem proving system [14] for details.

4. The type system of STT could be enriched with new machinery such as subtypes, dependent types, and user-defined type constructors.

5. A choice operator (ϵ) could be added to the set $\{@, \lambda, =, \mathrm{I}\}$ of primitive expression constructors of STT (see [14]).

6. The semantics of STT can be extended to include *partial functions* (like division and the limit of a function at a point) that are not defined on all arguments and *undefined expressions* (like $17/0$ and $\lim_{x \to 0} \sin \frac{1}{x}$) that do not denote any value. In particular, a definite description $(\mathrm{I}\, x : \alpha \ .\ A)$ would be undefined if there was not a unique value x of type α that satisfies A. This extension of the semantics can be done by either preserving the classical two-valued nature of simple type theory [9] or by extending simple type theory to a three-valued logic [21].

   Mathematics can be formalized more concisely and worked with more naturally in a logic in which partial functions and undefined expressions can be directly represented than in a standard logic in which all functions are assumed to be total and all expressions must be denoting. (Many examples supporting this statement can be found in the theory library of the IMPS theorem proving system [11].)

Convenient versions of Church's type theory have been implemented in the computer theorem proving systems HOL [14], IMPS [11], Isabelle [27], ProofPower [22], PVS [26], and TPS [3]. Experience has shown that these implemented versions of Church's type theory are indeed effective for practical use. Hundreds of theories have been formulated and thousands of theorems have been proved using these logics in these theorem provers.

Virtue 7 *There are practical extensions of* STT *that can be effectively implemented.*

9 Conclusion

We have surveyed seven virtues of simple type theory. These virtues make simple type theory a very attractive general-purpose logic. One can argue that, as a logic for actual use—in science, engineering, and mathematics education and research—simple type theory is superior to first-order logic. It is much closer to mathematical practice than first-order logic, and as a result, mathematics expressed in simple type theory is more natural, more

concise, and easier to work with. The greatest virtue of simple type theory is that it is a logic that is effective for *practice* as well as for *theory*.

We recommend that simple type theory be incorporated into introductory logic courses offered by mathematics departments by replacing the traditional two-logic sequence with a three-logic sequence—*propositional logic, first-order logic, simple type theory*. This is feasible since nearly all the basic ideas and principles of simple type theory are already found in first-order logic (the notion of a type is the most notable exception). After a shortened introduction to first-order logic, students could immediately proceed to simple type theory. Subjects, such as proof systems and model theory, could be briefly addressed in first-order logic and then explored further in simple type theory.

Although students would spend less time on directly studying first-order logic, their study of simple type theory would foster a deeper understanding of both first-order logic and the logical principles that transcend individual logics. But the most important benefit would be that the students—especially those who are going to become engineers and computer scientists—would go into the real world with a practical logic in their toolkit.

10 Acknowledgments

Special thanks is given to Peter Andrews for offering the author many useful suggestions and comments and for writing *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof* [2]. Many of the definitions associated with STT are modifications of definitions given in this excellent textbook. The author is also grateful to Wolfram Kahl, Jérémie Wajs, and Jeffrey Zucker for reading preliminary drafts of the paper.

References

[1] P. B. Andrews. A reduction of the axioms for the theory of propositional types. *Fundamenta Mathematicae*, 52:345–350, 1963.

[2] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof, Second Edition*. Kluwer, 2002.

[3] P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfennig, and H. Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16:321–353, 1996.

[4] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[5] L. Chwistek. Antynomje logikiformalnej. *Przeglad Filozoficzny*, 24:164–171, 1921.

[6] R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[7] Coq Development Team. *The Coq Proof Assistant Reference Manual, Version 7.4*, 2003. Available at `http://pauillac.inria.fr/coq/doc/main.html`.

[8] R. Dedekind. Was sind und was sollen die Zahlen? (1888). In W. W. Beman, editor, *Essays on the Theory of Numbers*. Dover, 2001. Translated into English by the editor.

[9] W. M. Farmer. A partial functions version of Church's simple theory of types. *Journal of Symbolic Logic*, 55:1269–91, 1990.

[10] W. M. Farmer. A basic extended simple type theory. SQRL Report No. 14, McMaster University, 2003.

[11] W. M. Farmer, J. D. Guttman, and F. J. Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11:213–248, 1993.

[12] S. Feferman. Predicativity. In S. Shapiro, editor, *Handbook of the Philosophy of Mathematics and Logic*. Oxford University Press. Forthcoming.

[13] R. O. Gandy. The simple theory of types. In R. Gandy and M. Hyland, editors, *Logic Colloquium 76*, pages 173–181. North-Holland, 1977.

[14] M. J. C. Gordon and T. F. Melham. *Introduction to* HOL*: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[15] A. P. Hazen. Predicative logics. In D. M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume I, pages 331–407. Reidel, 1983.

[16] L. Henkin. The completeness of the first-order functional calculus. *Journal of Symbolic Logic*, 15:159–166, 1949.

[17] L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15:81–91, 1950.

[18] L. Henkin. A theory of propositional types. *Fundamenta Mathematicae*, 52:323–344, 1963.

[19] L. Henkin. The discovery of my completeness proofs. *Bulletin of Symbolic Logic*, 2:127–158, 1996.

[20] R. B. Jensen. On the consistency of a slight (?) modification of Quine's NF. *Synthese*, 19:250–263, 1969.

[21] M. Kerber and M. Kohlhase. A mechanization of strong Kleene logic for partial functions. In A. Bundy, editor, *Automated Deduction—CADE-12*, volume 814 of *Lecture Notes in Computer Science*, pages 371–385. Springer-Verlag, 1994.

[22] Lemma 1 Ltd. *ProofPower: Description*, 2000. Available at `http://www.lemma-one.com/ProofPower/doc/doc.html`.

[23] S. Mac Lane. *Mathematics: Form and Function*. Springer-Verlag, 1986.

[24] A. R. D. Mathias. The strength of Mac Lane set theory. *Annals of Pure and Applied Logic*, 110:107–234, 2001.

[25] R. P. Nederpelt, J. H. Geuvers, and R. C. De Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic and The Foundations of Mathematics*. North Holland, 1994.

[26] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. A. Henzinger, editors, *Computer Aided Verification: 8th International Conference, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414. Springer-Verlag, 1996.

[27] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[28] G. Peano. *Arithmetices principia nova methodo exposita*. Bocca, Turin, Italy, 1889.

[29] Robert Pollack. *The Theory of* LEGO. PhD thesis, University of Ediburgh, 1994.

[30] F. P. Ramsey. The foundations of mathematics. *Proceedings of the London Mathematical Society, Series 2*, 25:338–384, 1926.

[31] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908.

[32] A. M. Turing. Practical forms of type theory. *Journal of Symbolic Logic*, 13:80–94, 1948.

[33] A. N. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, 1910–13. Paperback version to section *56 published in 1964.