STMM: A Set Theory for Mechanized Mathematics[*]

William M. Farmer (`wmfarmer@mcmaster.ca`)
Department of Computing and Software
McMaster University
1280 Main Street West
Hamilton, Ontario, Canada L8S 4L7

5 October 2000

Abstract. Although set theory is the most popular foundation for mathematics, not many *mechanized mathematics systems* are based on set theory. Zermelo-Fraenkel (ZF) set theory and other traditional set theories are not an adequate foundation for mechanized mathematics. STMM is a version of von-Neumann-Bernays-Gödel (NBG) set theory that is intended to be a Set Theory for Mechanized Mathematics. STMM allows terms to denote proper classes and to be undefined, has a definite description operator, provides a sort system for classifying terms by value, and includes lambda-notation with term constructors for function application and function abstraction. This paper describes STMM and discusses why it is a good foundation for mechanized mathematics.

Keywords: Set theory, NBG, higher-order logic, mechanized mathematics, theorem proving systems, partial functions, undefinedness, sorts.

1. Introduction

The mathematics process consists of formulating mathematical models and then exploring them by stating and proving conjectures and by performing calculations. The goal of *mechanized mathematics* is to produce computer systems that support and improve the mathematics process. Today *mechanized mathematics systems* come in two major types. *Computer algebra systems*[1] enable the user to perform many kinds of calculations, both symbolic and numeric. *Theorem proving systems*[2] assist the user in developing mathematical models in the form of axiomatic theories. The development is guided and checked by formally proving that certain conjectures are theorems of the theory being developed.

Set theory has been the most popular foundation for mathematics for the better part of the 20th century. Consequently, mathematicians understand set theory quite well, and all serious mathematics practitioners are familiar with it to some degree. Although it is based on the two simple notions of set and membership, it is extremely expressive:

[*] Published in the *Journal of Automated Reasoning*, 26:269–289, 2001.

nearly all mathematical concepts can be expressed in terms of set and membership. A large part of the success of set theory is due to there being a standard formalization known as Zermelo-Fraenkel (ZF) set theory.

Set theory has not been a very popular foundation for mechanized mathematics. Only a few mechanized mathematics systems are based on set theory. These include the following:

1. The EVES program verification system [10] based on ZF.

2. M. Gordon's augmentation of HOL [23] with ZF axioms [22].

3. N. Megill's Metamath proof verifier [27] based on ZF.

4. The Mizar proof development system [38] based on Tarski-Grothendieck set theory.[3]

5. L. Paulson's implementation of ZF [34] in the Isabelle generic theorem prover [35].

6. A. Quaife's clausal formalization of von-Neumann-Bernays-Gödel (NBG) set theory [36] in the Otter resolution theorem prover [26].

What is wrong with set theory as foundation for mechanized mathematics? Although classical set theories such as ZF and NBG have great expressive power, they lack the practical machinery needed for mechanized mathematics. In particular, they do not provide built-in support for reasoning with functions and classifying terms by value (as type theory does).

Is there a version of set theory that would be a suitable foundation for mechanized mathematics? This paper addresses this question. It gives a set of design goals that we would like a set theory for mechanized mathematics to satisfy. We explain why ZF and NBG do not satisfy all of these goals. We then propose a new Set Theory for Mechanized Mathematics called STMM[4] that does satisfy each of the design goals. After defining STMM, we discuss the support STMM offers for functions, issues concerning the implementation of STMM, and how STMM compares with other implemented set theories. The paper ends with a short conclusion.

2. Design Goals

The following are six design goals for a set theory intended to serve as a logical foundation for mechanized mathematics:

Design Goal 1 (DG1). *The set theory includes the standard machinery of predicate logic as well as the machinery of set theory.*

In a set theory with the standard machinery of predicate logic, assertions about sets can be made that involve quantification and the application of predicates and operators. ZF, NBG, and related set theories satisfy this first goal.

Design Goal 2 (DG2) *Terms may denote both sets (small collections) and proper classes (big collections).*

Many basic mathematical objects, such as the universe of sets and the mapping that takes a set to its cardinality, are proper classes. In ZF, terms range over sets but not proper classes, while in NBG, terms range over both sets and proper classes. (Proper classes are usually represented in ZF by predicate symbols—which are not terms.) Thus, ZF does not satisfy this goal, but NBG does.

Design Goal 3 (DG3) *Terms may be undefined.*

Undefined terms, like x/x where $x = 0$ and $\lim_{x\to\infty} \sin x$, are a common and unavoidable part of mathematics. They arise naturally in mathematics practice from the use of partial functions and definite descriptions. The direct use of undefined terms facilitates efficient, compact expression of many mathematical ideas. This is the reason that undefined terms are commonly employed in informal mathematics.

In most set theories, including ZF and NBG, terms are not allowed to be undefined since the underlying logic is usually standard first-order logic. There are many approaches for dealing with partial functions and undefinedness in formal logic. The major approaches are discussed in [11].

Design Goal 4 (DG4) *There is a definite description operator for forming terms like the "the unique x that satisfies the property P".*

Definite description is a basic tool for defining mathematical concepts. It is employed in some fashion or another throughout mathematics. Set theories like ZF and NBG do not normally contain a definite description operator. Definite descriptions are awkward in a system in which all terms must be defined because a nondenoting definite description like "the unique x such $x \neq x$" must denote some object.

Design Goal 5 (DG5) *There is a built-in system for classifying terms by value (as there is in type theory).*

A term classification system is very useful for providing some immediate information about the value of a term (e.g., that the term denotes some integer) before the value of term is known itself (e.g., that the term denotes 17).

There is no built-in system for classifying terms by value in either ZF or NBG. In particular, ZF terms are an inadequate classification system because they denote sets, but not proper classes. Thus, for example, there is no term that can be a classification for the domain of all terms or a classification for the domain of terms that denote functions. Moreover, ZF terms may denote the empty set. Allowing classifications that may be empty would severely complicate a classification system for terms. Such terms are hard to avoid since it is undecidable whether an arbitrary ZF term denotes the empty set.

Many mechanized mathematics systems are based on type theory. Type theory provides a system for classifying terms by value, but it also clashes with mathematics practice [25]. For example, in type theory there are different operators for different types. When two types have similar structure, some of their operators may be similar to each other. For instance, an identity function on one type will behave the same as an identity function on another type. To deal with distinct operators with similar behavior, many type theories admit "polymorphic" operators that may be applied to arguments of different types. Polymorphic operators are seldom employed in mathematics practice since sets, and not types, dominate mathematics. In a set theory like ZF or NBG, there is just one type (sets or classes), so there is no need for polymorphic operators.

Design Goal 6 (DG6) *There is good support for functions including lambda-notation (or an equivalent notation) with term constructors for function application and function abstraction.*

Functions, both partial and total, are just as basic to mathematics as sets and need to be fully supported.

ZF has poor support for functions. The set-theoretic machinery in ZF is inadequate. Since terms denote sets but not proper classes, terms can represent some partial functions but not any total functions. Furthermore, there are no term constructors for function application and function abstraction as with lambda-notation. It would be awkward to add function application and abstraction term constructors to a system like ZF in which all terms are defined. Since terms can represent only partial functions, many applications of a term denoting a function would be nondenoting.

The (first-order) logical machinery in ZF is also inadequate. Operator symbols represent functions, but only total functions. Moreover, operator symbols are not terms. There is a term constructor for operator application but not for operator abstraction.

In summary, ZF satisfies just **DG1**, and NBG satisfies only **DG1** and **DG2**.

3. Overview of STMM

STMM is a version of von-Neumann-Bernays-Gödel (NBG) set theory. As we shall see, STMM satisfies all six of the design goals given in the preceding section.

In contrast to ZF, variables in NBG range over both sets and proper classes. Thus, the universe of sets V and total functions from V to V like the cardinality function can be defined as terms in NBG even though they are proper classes. (A good introduction to NBG is found in [28].)

NBG is closely related to ZF. The underlying logic of both NBG and ZF is first-order logic, and NBG and ZF both share the same intuitive model of the iterated hierarchy of sets. The nonlogical axioms of NBG are very similar to those of ZF; most of them are simply ZF axioms with some of the quantifiers restricted to sets. And there is a faithful interpretation of ZF in NBG [32, 37, 40], which implies that ZF is consistent iff NBG is consistent. However, NBG is finitely axiomatizable ([21] and [28] present finite axiomatizations of NBG), while ZF is not (see [24] or [30] for a proof).

As a version of NBG, STMM satisfies **DG1** and **DG2** and has the same expressive power as NBG.

STMM has additional machinery that makes it much more appropriate for mechanized mathematics than ordinary NBG. The additional machinery includes:

1. Support for undefined terms (so **DG3** is satisfied).
2. A definite description operator (so **DG4** is satisfied).
3. A system for classifying terms by value (so **DG5** is satisfied).
4. Support for (partial and total) functions (so **DG6** is satisfied).

This machinery has the the same flavor as the special machinery of LUTINS [11, 12, 13], the logic of the IMPS Interactive Mathematical Proof System [20, 17]. LUTINS closely corresponds to mathematics practice and has proven to be an effective logic for formalizing traditional mathematics (e.g., see [16]).

The following are the major ingredients of STMM. In contrast to ZF and NBG, the underlying logic of STMM is Partial First-Order Logic (PFOL), a version of first-order logic that admits partial functions and undefined terms (see [14, 15]). STMM contains the usual vocabulary and axioms of NBG. It also contains a "sort system" for classifying terms by value that is similar to the sort system of LUTINS. Lastly, STMM has term constructors for function application and function abstraction—which provides STMM with lambda-notation for reasoning with functions.

The ingredients of STMM are briefly discussed in the remainder of this section, and then STMM is defined in the next section.

3.1. PFOL

PFOL is a formalization of what we call the traditional approach to partial functions [14]. PFOL is closely related to the partial logics proposed by R. Schock [39], T. Burge [6, 7], M. Beeson [3, 4], and L. Monk [29].

The syntax of PFOL is very similar to the syntax of ordinary first-order logic. PFOL has the usual logical connectives $(=, \neg, \wedge, \vee, \supset, \equiv, \forall, \exists)$ plus a *definite description operator* I. I is used to construct *definite descriptions*, that is, terms of the form $(\mathrm{I}\, x \;.\; \varphi)$. A term $(\mathrm{I}\, x \;.\; \varphi)$ denotes the unique x that satisfies φ if there is such an x and is undefined otherwise.

The semantics of PFOL is a modification of the semantics of ordinary first-order logic. Although individual constants and predicate symbols have the same semantics as usual, operator symbols are allowed to denote partial functions. Since definite descriptions and applications of operator symbols may be undefined, the valuation function on terms is partial, but the valuation function on formulas is total. In particular, if p is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms, then the application $p(t_1, \ldots, t_n)$ will be *false* if any argument t_i is undefined.

3.2. Classifying Terms

A *sort* is a syntactic object intended to denote a nonempty domain of values. In the case of STMM, these values are actually classes. Semantically, a sort is just a nonempty unary predicate. Like types, sorts are assigned to terms on the basis of their syntax, but unlike types, sorts may be assigned to nondenoting terms (i.e., terms that are not "type correct"). If a term t is assigned a sort α, then the value of t is a member of the domain denoted by α *provided t is defined.* (Whether or not a term is defined is an undecidable problem.) Thus, sorts provide a means to classify terms by value that does not depend on whether the terms are defined or undefined.

3.3. Support for Functions

STMM provides especially good support for working with partial and total functions. As in ZF and NBG, partial functions from sets to sets are represented in STMM as certain sets of ordered pairs. However, unlike ZF and NBG, STMM has special machinery—lambda-notation and sorts—for directly manipulating terms that denote partial and total functions and for keeping track of when an application of a function is defined and what kind of value an application of a function may have when it is defined.

The special machinery in STMM for reasoning with functions is discussed in more detail in section 5.

3.4. Relationship to NBG

An alternate version of STMM, called NBG*, is described in [14] and defined in [15]. NBG* is defined in stages, while STMM is defined in a direct way. The purpose of NBG* is to illustrate precisely how it (and STMM) is related to NBG. It is for study, not use. STMM, on the other hand, is intended to be a logic that can actually be implemented and used as a component of a mechanized mathematics system.

STMM and NBG* have the expressive power of NBG plus type-theoretic like machinery. The theorem below, which follows from the results in [15], shows that this additional machinery is purely a convenience.

THEOREM 3.1. *For every theory $\mathcal{T}$ of* NBG**, there is a theory of ordinary* NBG $\mathcal{T}^*$ *and a computable translation from each formula φ of $\mathcal{T}$ to a formula φ^* of $\mathcal{T}^*$ such that*

$$\mathcal{T} \models \varphi \;\; \textit{iff} \;\; \mathcal{T}^* \models \varphi^*.$$

4. The Definition of STMM

4.1. Sort Systems

A *sort* is a syntactic object intended to denote a nonempty domain of values. Sorts are used to classify terms by value.

The set of sort symbols built from a set S of atomic sort symbols will be the set $\Omega(S)$ defined below.

Let S be a set of symbols. $\Omega(S)$ is the set defined by:

1. $S \subseteq \Omega(S)$.
2. If $\alpha, \beta \in \Omega(S)$, then $\alpha \rightharpoonup \beta \in \Omega(S)$.

A sort of the form $\alpha \rightharpoonup \beta$ is intended to denote the domain of partial functions from the domain denoted by α to the domain denoted by β.

Given a total function $f : S \to \Omega(S)$, $\preceq_f$ is the smallest binary relation on $\Omega(S)$ such that:

1. If $\alpha \in S$, then $\alpha \preceq_f f(\alpha)$.
2. $\preceq_f$ is reflexive, i.e., for all $\alpha \in \Omega(S)$, $\alpha \preceq_f \alpha$.
3. $\preceq_f$ is transitive, i.e., for all $\alpha, \beta, \gamma \in \Omega(S)$, if $\alpha \preceq_f \beta$ and $\beta \preceq_f \gamma$, then $\alpha \preceq_f \gamma$.
4. If $\alpha_1 \preceq_f \alpha_2$ and $\beta_1 \preceq_f \beta_2$, then $\alpha_1 \rightharpoonup \beta_1 \preceq_f \alpha_2 \rightharpoonup \beta_2$.

The intended meaning of $\alpha \preceq_f \beta$ is that the domain denoted by α is included in the domain denoted by β.

$\preceq_f$ is *noetherian* if every ascending sequence of members of $\Omega(S)$,

$$\alpha_1 \preceq_f \alpha_2 \preceq_f \alpha_3 \preceq_f \cdots,$$

is eventually stationary, i.e., there is some m such that $\alpha_i = \alpha_m$ for all $i \geq m$. If $\preceq_f$ is noetherian, then $\preceq_f$ is obviously antisymmetric (i.e., for all $\alpha, \beta \in \Omega(S)$, if $\alpha \preceq_f \beta$ and $\beta \preceq_f \alpha$, then $\alpha = \beta$). Hence, $\preceq_f$ is a partial order if it is noetherian.

A *sort system* of STMM is a pair $(\mathcal{A}, \xi)$ where $\mathcal{A}$ is a set of symbols with $\mathbf{C}, \mathbf{V} \in \mathcal{A}$ and ξ is a total function from $\mathcal{A}$ to $\Omega(\mathcal{A})$ such that:

1. For all $\alpha \in \mathcal{A}$, $\xi(\alpha) = \alpha$ iff $\alpha = \mathbf{C}$.
2. $\preceq_\xi$ is noetherian.

C and **V** are intended to denote the domains of classes and sets, respectively. The relation $\preceq_\xi$ is not an absolutely necessary part of a sort system. It is included for the sake of convenience; we will see later the benefits that it offers.

A *sort* of $(\mathcal{A}, \xi)$ is any member of $\Omega(\mathcal{A})$. The sorts in $\mathcal{A}$ and $\Omega(\mathcal{A}) \setminus \mathcal{A}$ are called *atomic sorts* and *compound sorts*, respectively. The *enclosing sort* of $\alpha \in \mathcal{A}$ is the sort $\xi(\alpha)$. The *least upper bound* of α and β, written $\alpha \sqcup_\xi \beta$, is the least upper bound of α and β in the partial order $\preceq_\xi$. (The least upper bound of two sorts is not always defined.) The maximal sorts in $\preceq_\xi$ are $\mathbf{C}$ plus the compound sorts formed from $\mathbf{C}$ alone.

A *function sort* is either a compound sort or an atomic sort α such that $\alpha \preceq_\xi \beta$ for some compound sort $\beta \in \Omega(\mathcal{A})$. A function sort, whether atomic or compound, is intended to denote a domain of partial functions. The *range sort* of a function sort α, written $\mathrm{ran}(\alpha)$, is defined as follows: if $\alpha = \beta \rightharpoonup \gamma$, then $\mathrm{ran}(\alpha) = \gamma$; otherwise, $\mathrm{ran}(\alpha) = \mathrm{ran}(\xi(\alpha))$. (We leave it as an exercise to the reader to verify that the notion of a range sort is well defined.)

A *sort frame* for $(\mathcal{A}, \xi)$ is a set $\{\mathcal{D}_\alpha : \alpha \in \Omega(\mathcal{A})\}$ of nonempty domains such that:

1. If $\alpha \preceq_\xi \beta$, then $\mathcal{D}_\alpha \subseteq \mathcal{D}_\beta$.

2. $\mathcal{D}_\alpha \subseteq \mathcal{D}_{\mathbf{C}}$ for all $\alpha \in \Omega(\mathcal{A})$.

A sort frame is a "model" for a sort system. The relation $\preceq_\xi$ is used to place a minimal structure on the members of a sort frame.

4.2. Syntax

A *variable* of STMM is a member of a fixed infinite set $\mathcal{V}$ of symbols. A *language* of STMM is a tuple $(\mathcal{C}, \mathcal{O}, \mathcal{P}, \mathcal{A}, \xi, \sigma)$ such that:

1. $\mathcal{C}$ is a set of *individual constants.*

2. $\mathcal{O}$ is a set of *operator symbols*, each with an assigned arity ≥ 1. $\mathcal{O}$ contains the binary operator symbol ordered-pair.

3. $\mathcal{P}$ is a set of *predicate symbols*, each with an assigned arity ≥ 1. $\mathcal{P}$ contains the unary predicate symbol function and the binary predicate symbols $=$ and $\in$.

4. $(\mathcal{A}, \xi)$ is a sort system.

5. $\sigma : \mathcal{V} \cup \mathcal{C} \to \Omega(\mathcal{A})$ is total function such that
$$\mathcal{V}_\alpha = \{x \in \mathcal{V} : \sigma(x) = \alpha\}$$
is infinite for all $\alpha \in \Omega(\mathcal{A})$.

6. $\mathcal{V}$, $\mathcal{C}$, $\mathcal{O}$, $\mathcal{P}$, and $\mathcal{A}$ are pairwise disjoint.

In the remainder of this paper, let $\mathcal{L} = (\mathcal{C}, \mathcal{O}, \mathcal{P}, \mathcal{A}, \xi, \sigma)$ be a language of STMM. $\mathcal{L}$ is *minimal* if $\mathcal{C} = \emptyset$, $\mathcal{O} = \{\mathsf{ordered\text{-}pair}\}$, $\mathcal{P} = \{\mathsf{function}, =, \in\}$, and $\mathcal{A} = \{\mathbf{C}, \mathbf{V}\}$.

Let $\mathcal{L}_1 = (\mathcal{C}_1, \mathcal{O}_1, \mathcal{P}_1, \mathcal{A}_1, \xi_1, \sigma_1)$ and $\mathcal{L}_2 = (\mathcal{C}_2, \mathcal{O}_2, \mathcal{P}_2, \mathcal{A}_2, \xi_2, \sigma_2)$ be languages of STMM. $\mathcal{L}_1$ is a *sublanguage* of $\mathcal{L}_2$ if $\mathcal{C}_1 \subseteq \mathcal{C}_2$, $\mathcal{O}_1 \subseteq \mathcal{O}_2$, $\mathcal{P}_1 \subseteq \mathcal{P}_2$, $\mathcal{A}_1 \subseteq \mathcal{A}_2$, ξ_1 is a subfunction of ξ_2, and σ_1 is a subfunction of σ_2.

A *term of sort* α of $\mathcal{L}$ and a *formula* of $\mathcal{L}$ are simultaneously defined by the rules below. $\mathbf{var}_{\mathcal{L}}[x, \alpha]$ asserts that $x \in \mathcal{V}$ and $\alpha = \sigma(x)$, $\mathbf{term}_{\mathcal{L}}[t, \alpha]$ asserts that t is a term of sort α of $\mathcal{L}$, and $\mathbf{form}_{\mathcal{L}}[\varphi]$ asserts that φ is a formula of $\mathcal{L}$.

$$\mathbf{T1}\quad \frac{a \in \mathcal{V} \cup \mathcal{C}}{\mathbf{term}_{\mathcal{L}}[a, \sigma(a)]}$$

$$\mathbf{T2}\quad \frac{\mathbf{var}_{\mathcal{L}}[x, \alpha],\ \mathbf{form}_{\mathcal{L}}[\varphi]}{\mathbf{term}_{\mathcal{L}}[(\mathrm{I}\, x : \alpha \,.\, \varphi), \alpha]}$$

$$\mathbf{T3}\quad \frac{\mathbf{form}_{\mathcal{L}}[\varphi],\ \mathbf{term}_{\mathcal{L}}[s, \alpha],\ \mathbf{term}_{\mathcal{L}}[t, \beta]}{\mathbf{term}_{\mathcal{L}}[\mathsf{if}(\varphi, s, t), \gamma]}$$

$$\text{where } \gamma = \begin{cases} \alpha \sqcup_{\xi} \beta & \text{if } \alpha \sqcup_{\xi} \beta \text{ is defined} \\ \mathbf{C} & \text{otherwise} \end{cases}$$

$$\mathbf{T4}\quad \frac{\mathbf{term}_{\mathcal{L}}[f, \alpha],\ \mathbf{term}_{\mathcal{L}}[a, \beta]}{\mathbf{term}_{\mathcal{L}}[f[a], \gamma]}$$

$$\text{where } \gamma = \begin{cases} \mathrm{ran}(\alpha) & \text{if } \alpha \text{ is a function sort} \\ \mathbf{V} & \text{otherwise} \end{cases}$$

$$\mathbf{T5}\quad \frac{\mathbf{var}_{\mathcal{L}}[x, \alpha],\ \mathbf{term}_{\mathcal{L}}[t, \beta],}{\mathbf{term}_{\mathcal{L}}[(\lambda\, x : \alpha \,.\, t), \alpha \rightharpoonup \beta]}$$

$$\mathbf{T6}\quad \frac{o \in \mathcal{O} \text{ is } n\text{-ary},\ \mathbf{term}_{\mathcal{L}}[t_1, \alpha_1], \ldots, \mathbf{term}_{\mathcal{L}}[t_n, \alpha_n]}{\mathbf{term}_{\mathcal{L}}[o(t_1, \ldots, t_n), \mathbf{C}]}$$

$$\mathbf{F1}\quad \frac{p \in \mathcal{P} \text{ is } n\text{-ary},\ \mathbf{term}_{\mathcal{L}}[t_1, \alpha_1], \ldots, \mathbf{term}_{\mathcal{L}}[t_n, \alpha_n]}{\mathbf{form}_{\mathcal{L}}[p(t_1, \ldots, t_n)]}$$

$$\textbf{F2}\quad \frac{\mathbf{form}_{\mathcal{L}}[\varphi]}{\mathbf{form}_{\mathcal{L}}[\neg\varphi]}$$

$$\textbf{F3}\quad \frac{\mathbf{form}_{\mathcal{L}}[\varphi],\ \mathbf{form}_{\mathcal{L}}[\psi]}{\mathbf{form}_{\mathcal{L}}[(\varphi \supset \psi)]}$$

$$\textbf{F4}\quad \frac{\mathbf{var}_{\mathcal{L}}[x,\alpha],\ \mathbf{form}_{\mathcal{L}}[\varphi]}{\mathbf{form}_{\mathcal{L}}[(\forall\, x : \alpha \,.\, \varphi)]}$$

When t is a term, let $\bar{\sigma}(t)$ denote the sort of t. A term of the form $(\mathrm{I}\, x : \alpha \,.\, \varphi)$, $\mathsf{if}(\varphi, s, t)$, $f[a]$, or $(\lambda\, x : \alpha \,.\, t)$ is called a *definite description*, *conditional term*, *function application*, or *function abstraction*, respectively. Notice that the relation $\preceq_\xi$ enables nontrivial sorts to be assigned to conditional terms and function applications in certain cases.

Parentheses in terms and formulas may be suppressed when meaning is not lost. For convenience, we also employ the following abbreviations:

$(s = t)$	for	$= (s, t)$.
$(s \neq t)$	for	$\neg(s = t)$.
$(s \in t)$	for	$\in (s, t)$.
$(s \notin t)$	for	$\neg(s \in t)$.
$\langle s, t \rangle$	for	$\mathsf{ordered\text{-}pair}(s, t)$.
$(\varphi \wedge \psi)$	for	$\neg(\varphi \supset \neg\psi)$.
$(\varphi \vee \psi)$	for	$\neg\varphi \supset \psi$.
$(\varphi \equiv \psi)$	for	$(\varphi \supset \psi) \wedge (\psi \supset \varphi)$.
$(\exists\, x : \alpha \,.\, \varphi)$	for	$\neg(\forall\, x : \alpha \,.\, \neg\varphi)$.
$\square\, x_1, \ldots, x_n : \alpha \,.\, \varphi$	for	$\square\, x_1 : \alpha \ldots \square\, x_n : \alpha \,.\, \varphi$ where $\square$ is $\forall$ or $\exists$.
$(t \downarrow \alpha)$	for	$\exists\, x : \alpha \,.\, x = t$ where x does not occur in t.
$(t\downarrow)$	for	$(t \downarrow \bar{\sigma}(t))$.
$(t\uparrow)$	for	$\neg(t\downarrow)$.
$(s \simeq t)$	for	$(s\downarrow \vee\, t\downarrow) \supset s = t$.
$\bot_\alpha$	for	$\mathrm{I}\, x : \alpha \,.\, x \neq x$.
$(\alpha \ll \beta)$	for	$\forall\, x : \alpha \,.\, (x \downarrow \beta)$.

$(t \downarrow \alpha)$, $t\downarrow$, $t\uparrow$, and $s \simeq t$ are read as "t is defined in α", "t is defined", "t is undefined", and "s and t are quasi-equal". $\bot_\alpha$ is a canonical undefined term of sort α, and $(\alpha \ll \beta)$ says α is a subsort of β.

Let an *expression* of $\mathcal{L}$ be either a term or formula of $\mathcal{L}$. "Free variable", "closed", and similar notions are defined in the obvious way. A *sentence* is a closed formula.

4.3. Semantics

We shall present the semantics of STMM in two stages. In the first stage, we introduce the "logic" portion of the semantics embodied in the notion of a model of a STMM language. The logic portion of STMM is Partial First-Order Logic (PFOL), which we discussed in section 3. In the second stage, we formulate the "set theory" portion of the semantics by expressing the axioms of NBG set theory as STMM sentences and then defining a STMM theory to be a STMM language plus a set of sentences that includes these axioms.

A *structure* for $\mathcal{L}$ is a pair $(\{\mathcal{D}_\alpha : \alpha \in \Omega(\mathcal{A})\}, I)$ where $\{\mathcal{D}_\alpha : \alpha \in \Omega(\mathcal{A})\}$ is a sort frame for $(\mathcal{A}, \xi)$ and I is a total function on $\mathcal{C} \cup \mathcal{O} \cup \mathcal{P}$ such that:

1. If $a \in \mathcal{C}$, $I(a) \in \mathcal{D}_{\sigma(a)}$.
2. If $o \in \mathcal{O}$ is n-ary, $I(o)$ is a partial function from $\mathcal{D}_{\mathbf{C}} \times \cdots \times \mathcal{D}_{\mathbf{C}}$ (n times) to $\mathcal{D}_{\mathbf{C}}$.
3. If $p \in \mathcal{P}$ is n-ary, $I(p)$ is a total function from $\mathcal{D}_{\mathbf{C}} \times \cdots \times \mathcal{D}_{\mathbf{C}}$ (n times) to $\{\text{T}, \text{F}\}$ (the domain of truth values). $I(=)$ is the identity relation on $\mathcal{D}_{\mathbf{C}}$.

Let $\mathcal{M} = (\{\mathcal{D}_\alpha : \alpha \in \Omega(\mathcal{A})\}, I)$ be a structure for $\mathcal{L}$. A *variable assignment* into $\mathcal{M}$ is a function that maps each $x \in \mathcal{V}$ to an element of $\mathcal{D}_{\sigma(x)}$. Given a variable assignment A into $\mathcal{M}$, $x \in \mathcal{V}$, and $d \in \mathcal{D}_{\sigma(x)}$, let $A[x \mapsto d]$ be the variable assignment A' into $\mathcal{M}$ such $A'(x) = d$ and $A'(y) = A(y)$ for all $y \neq x$.

Define $V = V^{\mathcal{M}}$ to be the binary function such that the following conditions are satisfied for all variable assignments A into $\mathcal{M}$ and all expressions of $\mathcal{L}$:

1. If $t \in \mathcal{V}$, then $V_A(t) = A(t)$.
2. If $t \in \mathcal{C}$, then $V_A(t) = I(t)$.
3. Let $t = (\mathrm{I}\, x : \alpha \mathrel{.} \varphi)$. If there is a unique $d \in \mathcal{D}_\alpha$ such that $V_{A[x \mapsto d]}(\varphi) = \text{T}$, then $V_A(t) = d$; otherwise $V_A(t)$ is undefined.
4. Let $t = \mathsf{if}(\varphi, t_1, t_2)$, $\gamma = \bar{\sigma}(t)$, and
$$t' = \mathrm{I}\, x : \gamma \mathrel{.} (\varphi \supset x = t_1) \wedge (\neg\varphi \supset x = t_2)$$
where x does not occur in φ, t_1, or t_2. If $V_A(t')$ is defined, then $V_A(t) = V_A(t')$; otherwise $V_A(t)$ is undefined.

5. Let $t = f[a]$, $\beta = \bar{\sigma}(t)$, and

$$t' = \mathrm{I}\, b : \beta \;.\; \mathsf{function}(f) \wedge \langle a, b \rangle \in f$$

where b does not occur in f or a. If $V_A(t')$ is defined, then $V_A(t) = V_A(t')$; otherwise $V_A(t)$ is undefined.

6. Let $t = (\lambda\, x : \alpha \;.\; s)$, $\beta = \bar{\sigma}(s)$, and

$$t' = \mathrm{I}\, g : \alpha \rightharpoonup \beta \;.\; \forall\, x : \alpha \;.\; \mathsf{if}((x \downarrow \mathbf{V}) \wedge (s \downarrow \mathbf{V}), g[x] = s, g[x]\uparrow)$$

where g does not occur in s. If $V_A(t')$ is defined, then $V_A(t) = V_A(t')$; otherwise $V_A(t)$ is undefined.

7. Let $t = o(t_1, \ldots, t_n)$. If $V_A(t_1), \ldots, V_A(t_n)$ are defined and $I(o)$ is defined at $\langle V_A(t_1), \ldots, V_A(t_n)\rangle$, then $V_A(t) = I(o)(V_A(t_1), \ldots, V_A(t_n))$; otherwise $V_A(t)$ is undefined.

8. Let $\varphi = p(t_1, \ldots, t_n)$. If $V_A(t_1), \ldots, V_A(t_n)$ are defined, then $V_A(\varphi) = I(p)(V_A(t_1), \ldots, V_A(t_n))$; otherwise $V_A(\varphi) = \mathrm{F}$.

9. Let $\varphi = \neg\varphi'$. If $V_A(\varphi') = \mathrm{F}$, then $V_A(\varphi) = \mathrm{T}$; otherwise $V_A(\varphi) = \mathrm{F}$.

10. Let $\varphi = (\varphi' \supset \varphi'')$. If $V_A(\varphi') = \mathrm{T}$ and $V_A(\varphi'') = \mathrm{F}$, then $V_A(\varphi) = \mathrm{F}$; otherwise $V_A(\varphi) = \mathrm{T}$.

11. Let $\varphi = (\forall\, x : \alpha \;.\; \varphi')$. If $V_{A[x \mapsto d]}(\varphi') = \mathrm{T}$ for all $d \in \mathcal{D}_\alpha$, then $V_A(\varphi) = \mathrm{T}$; otherwise $V_A(\varphi) = \mathrm{F}$.

For an expression E, $V_A^{\mathcal{M}}(E)$ is called the *value* of E in $\mathcal{M}$ with respect to A (when it is defined). Notice that $V_A^{\mathcal{M}}$ is a partial valuation function on terms but a total valuation function on formulas. A term t of $\mathcal{L}$ is *defined* in $\mathcal{M}$ with respect to A if its value $V_A^{\mathcal{M}}(t)$ is defined. A formula φ of $\mathcal{L}$ is *valid* in $\mathcal{M}$ if $V_A^{\mathcal{M}}(\varphi) = \mathrm{T}$ for every variable assignment A into $\mathcal{M}$. Notice that the valuation function on conditional terms, function applications, and function abstractions is defined in each case in terms of the valuation function on definite descriptions.

If a term is defined (i.e., $t\downarrow$ is valid), then t is defined in its assigned sort (i.e., $(t \downarrow \bar{\sigma}(t))$) by definition and is defined in $\mathbf{C}$ (i.e., $(t \downarrow \mathbf{C})$) since $\mathcal{D}_{\mathbf{C}}$ is the domain of classes, which includes every domain.

$\mathcal{M}$ is a *model* for $\mathcal{L}$ if, for all $\alpha, \beta \in \Omega(\mathcal{A})$,

$$\begin{aligned}\forall\, f : \mathbf{C} \;.\; & (f \downarrow \alpha \rightharpoonup \beta) \equiv \\ & (\mathsf{function}(f) \wedge (\forall\, a, b : \mathbf{C} \;.\; f[a] = b \supset (a \downarrow \alpha) \wedge (b \downarrow \beta)))\end{aligned}$$

is valid in $\mathcal{M}$. That is, a model is a structure such that each $\mathcal{D}_{\alpha \rightharpoonup \beta}$ is the domain of (partial) functions from $\mathcal{D}_\alpha$ to $\mathcal{D}_\beta$. One can construct a model for $\mathcal{L}$ by appealing to the fact that $\preceq_\xi$ is noetherian.

Space limitations prevent us from presenting an axiom system for the logic portion of the semantics of STMM. It would be very similar to the axiom system for PFOL given in [15].

4.4. NBG Axioms

Let $\mathcal{L}$ be minimal. We present now an axiomatization of NBG as a set of sentences of $\mathcal{L}$. Our axiomatization is very similar to the finite axiomatization of NBG given by K. Gödel in [21]. The axiomatization consists of 31 axioms. The first 15 are *definitions* that define (usually) new symbols in terms of old symbols. The remaining 16 axioms are the "proper" axioms of NBG. We use the variables a, b, c to denote variables of sort $\mathbf{C}$ and w, x, y, z to denote variables of sort $\mathbf{V}$.

The first axiom defines $\mathbf{V}$ to be the domain of sets. (Recall that a set is a class that is a member of some other class.)

NBG1 (Definition of V) $\forall\, x : \mathbf{C} \,.\, (x \downarrow \mathbf{V}) \equiv \exists\, y : \mathbf{C} \,.\, x \in y$.

The next fourteen axioms define the operator and predicate symbols of NBG:

NBG2 (Definition of Pair) $\forall\, a, b : \mathbf{C} \,.\, \{a, b\} \simeq$
$(\mathrm{I}\, x : \mathbf{V} \,.\, (a \downarrow \mathbf{V}) \wedge (b \downarrow \mathbf{V}) \wedge \forall\, y : \mathbf{V} \,.\, y \in x \equiv (y = a \vee y = b))$.

NBG3 (Definition of Singleton) $\forall\, a : \mathbf{C} \,.\, \{a\} \simeq \{a, a\}$.

NBG4 (Definition of Ordered Pair) $\forall\, a, b : \mathbf{C} \,.\, \langle a, b\rangle \simeq$
$\{\{a\}, \{a, b\}\}$.

NBG5 (Definition of Ordered Triple) $\forall\, a, b, c : \mathbf{C} \,.\, \langle a, b, c\rangle \simeq$
$\langle a, \langle b, c\rangle\rangle$.

NBG6 (Definition of Subset) $\forall\, a, b : \mathbf{C} \,.\, a \subseteq b \equiv$
$(\forall\, x : \mathbf{V} \,.\, x \in a \supset x \in b)$.

NBG7 (Definition of Proper Subset) $\forall\, a, b : \mathbf{C} \,.\, a \subset b \equiv$
$(a \subseteq b \wedge a \neq b)$.

NBG8 (Definition of Empty) $\forall\, a : \mathbf{C} \,.\, \mathsf{empty}(a) \equiv \forall\, x : \mathbf{V} \,.\, x \notin a$.

NBG9 (Definition of Univocal) $\forall\, a : \mathbf{C} \,.\, \mathsf{univocal}(a) \equiv$
$(\forall\, x, y, z : \mathbf{V} \,.\, (\langle x, y\rangle \in a \wedge \langle x, z\rangle \in a) \supset y = z)$.

NBG10 (Definition of Function) $\forall\, a : \mathbf{C} \,.\, \mathsf{function}(a) \equiv (\mathsf{univocal}(a) \wedge \forall\, x : \mathbf{V} \,.\, x \in a \equiv (\exists\, y, z : \mathbf{V} \,.\, x = \langle y, z \rangle))$.

NBG11 (Definition of Intersection) $\forall\, a, b : \mathbf{C} \,.\, a \cap b \simeq (\mathrm{I}\, c : \mathbf{C} \,.\, \forall\, x : \mathbf{V} \,.\, x \in c \equiv (x \in a \wedge x \in b))$.

NBG12 (Definition of Complement) $\forall\, a : \mathbf{C} \,.\, \overline{a} \simeq (\mathrm{I}\, b : \mathbf{C} \,.\, \forall\, x : \mathbf{V} \,.\, x \in b \equiv x \not\in a)$.

NBG13 (Definition of Domain) $\forall\, a : \mathbf{C} \,.\, \mathsf{domain}(a) \simeq (\mathrm{I}\, b : \mathbf{C} \,.\, \forall\, x : \mathbf{V} \,.\, x \in b \equiv (\exists\, y : \mathbf{V} \,.\, \langle x, y \rangle \in a))$.

NBG14 (Definition of Sum Class) $\forall\, a : \mathbf{C} \,.\, \mathsf{sum}(a) \simeq (\mathrm{I}\, b : \mathbf{C} \,.\, \forall\, x : \mathbf{V} \,.\, x \in b \equiv (\exists\, y : \mathbf{V} \,.\, x \in y \wedge y \in a))$.

NBG15 (Definition of Power Class) $\forall\, a : \mathbf{C} \,.\, \mathsf{power}(a) \simeq (\mathrm{I}\, b : \mathbf{C} \,.\, \forall\, x : \mathbf{V} \,.\, x \in b \equiv x \subseteq a)$.

The first two proper axioms of NBG are:

NBG16 (Extensionality) $\forall\, a, b : \mathbf{C} \,.\, (\forall\, x : \mathbf{V} \,.\, x \in a \equiv x \in b) \supset a = b$.

NBG17 (Pairing) $\forall\, x, y : \mathbf{V} \,.\, \{x, y\}\!\downarrow$.

The next group of axioms assert the existence of certain classes:

NBG18 (Membership Class) $\exists\, a : \mathbf{C} \,.\, \forall\, x, y : \mathbf{V} \,.\, \langle x, y \rangle \in a \equiv x \in y$.

NBG19 (Intersection) $\forall\, a, b : \mathbf{C} \,.\, (a \cap b)\!\downarrow$.

NBG20 (Complement) $\forall\, a : \mathbf{C} \,.\, \overline{a}\!\downarrow$.

NBG21 (Domain) $\forall\, a : \mathbf{C} \,.\, \mathsf{domain}(a)\!\downarrow$.

NBG22 (Direct Product) $\forall\, a : \mathbf{C} \,.\, \exists\, b : \mathbf{C} \,.\, \forall\, x, y : \mathbf{V} \,.\, \langle x, y \rangle \in b \equiv x \in a$.

NBG23 (Permutation 1) $\forall\, a : \mathbf{C} \,.\, \exists\, b : \mathbf{C} \,.\, \forall\, x, y : \mathbf{V} \,.\, \langle x, y \rangle \in b \equiv \langle y, x \rangle \in a$.

NBG24 (Permutation 2) $\forall\, a : \mathbf{C} \,.\, \exists\, b : \mathbf{C} \,.\, \forall\, x, y, z : \mathbf{V} \,.\, \langle x, y, z \rangle \in b \equiv \langle y, z, x \rangle \in a$.

NBG25 (Permutation 3) $\forall\, a : \mathbf{C} \,.\, \exists\, b : \mathbf{C} \,.\, \forall\, x, y, z : \mathbf{V} \,.$ $\langle x, y, z \rangle \in b \equiv \langle x, z, y \rangle \in a$.

The next four axioms assert the existence of certain sets:

NBG26 (Infinity) $\exists\, x : \mathbf{V} \,.\, \neg\mathsf{empty}(x) \wedge$ $(\forall\, y : \mathbf{V} \,.\, y \in x \supset (\exists z : \mathbf{V} \,.\, z \in x \wedge y \subset z))$.

NBG27 (Sum Set) $\forall\, x : \mathbf{V} \,.\, (\mathsf{sum}(x) \downarrow \mathbf{V})$.

NBG28 (Power Set) $\forall\, x : \mathbf{V} \,.\, (\mathsf{power}(x) \downarrow \mathbf{V})$.

NBG29 (Replacement) $\forall\, a : \mathbf{C} \,.\, \mathsf{univocal}(a) \supset$ $(\forall\, x : \mathbf{V} \,.\, \exists\, y : \mathbf{V} \,.\, \forall\, z : \mathbf{V} \,.\, z \in y \equiv (\exists\, w : \mathbf{V} \,.\, w \in x \wedge \langle w, z \rangle \in a))$.

The last two axioms are the axioms of foundation and global choice:

NBG30 (Foundation) $\forall\, a : \mathbf{C} \,.\, \neg\mathsf{empty}(a) \supset$ $(\exists\, x : \mathbf{V} \,.\, x \in a \wedge \forall\, y : \mathbf{V} \,.\, \neg(y \in x \wedge y \in a))$.

NBG31 (Global Choice) $\exists\, f : \mathbf{C} \,.\, \mathsf{function}(f) \wedge$ $(\forall\, x : \mathbf{V} \,.\, \neg\mathsf{empty}(x) \supset f[x] \in x)$.

Notes:

1. $\{a, b\}$ and $\langle a, b \rangle$ are undefined whenever a is not a set or b is not a set.

2. In Gödel's axiomatization of NBG in [21], $\langle x, y \rangle$ represents a mapping of y to x; in our axiomatization it represents a mapping of x to y.

3. Axiom NBG22 says that, for all classes a, the direct product $a \times V$ is a class, where V is the class of all sets.

4. It is an exercise in p. 164 of [28] that the first permutation axiom, NBG23, follows from axioms NBG21, NBG22, NBG24, and NBG25.

5. The axiom of foundation, NBG30, is dispensable and could be replaced with an "antifoundation" axiom (see [1]).

6. The axiom of global choice, NBG31, implies the axiom of local choice (as given, for example, in [28]).

4.5. Theories

Let $\mathcal{L}_0 = (\emptyset, \mathcal{O}_0, \mathcal{P}_0, \mathcal{A}_0, \xi_0, \sigma_0)$ be a STMM language such that:

1. $\mathcal{O}_0$ contains only the operator symbols defined in the axioms above.

2. $\mathcal{P}_0$ contains only $=$, $\in$, and the predicate symbols defined in the axioms above.

3. $\mathcal{A}_0 = \{\mathbf{C}, \mathbf{V}\}$.

Also let $\Gamma_0 = \{\text{NBG1}, \ldots, \text{NBG31}\}$.

A *theory* of STMM is a pair $\mathcal{T} = (\mathcal{L}, \Gamma)$ where $\mathcal{L}$ is a language of STMM such that $\mathcal{L}_0$ is a sublanguage of $\mathcal{L}$ and Γ is a set of sentences of $\mathcal{L}$ such that $\Gamma_0 \subseteq \Gamma$. The theory $\mathcal{T}_0 = (\mathcal{L}_0, \Gamma_0)$ is called the *kernel theory* of STMM. Let $\mathcal{T} = (\mathcal{L}, \Gamma)$ be a theory of STMM. A *model* for $\mathcal{T}$ is a model for $\mathcal{L}$ in which each $\varphi \in \Gamma$ is valid. A formula φ of $\mathcal{L}$ is *valid* in $\mathcal{T}$, written $\mathcal{T} \models \varphi$, if it is valid in every model for $\mathcal{T}$.

5. Support for Functions

In this section, "mapping" means an arbitrary function, while "function" means a particular kind of class that represents a function from sets to sets and "operator" means a function from classes to classes.

5.1. Functions

A *function* in STMM is a univocal class of ordered pairs that represents a partial or total mapping from sets to sets. If the domain of a function happens to be a set, the function itself is a set. If the domain is a proper class, the function is a proper class. A function that is a set is necessarily partial, while a total function is necessarily a proper class. A total function represents a mapping defined on all sets, such as the mapping that takes a set to its cardinality. Mappings defined on all sets arise naturally in mathematics practice. In STMM, they are first-class objects: they can be quantified over and represented by terms. They are not first-class objects in ZF and related set theories that do not admit proper classes.

5.2. Operators

An *operator* in STMM is a partial or total mapping from classes to classes. Operator symbols denote operators, but in general terms do not. Since an operator symbol is not a term, predicate symbols and other operator symbols cannot be applied to them. This dichotomy between terms denoting set mappings and operator symbols denoting class mappings reflects a natural division between the different roles mappings play in mathematics. A mapping that is to be *reasoned about*—such as a mapping from the real numbers to the real numbers—is usually a class; in STMM it can be the value of a fully endowed term. A mapping that is only to assist in *forming assertions*—such as the mapping that takes a function to its domain—may not be a class; in STMM it can be the value of an operator symbol that can be used to form terms but is not a term itself.

5.3. Sorts

A *sort* α of $\mathcal{L}$ is a syntactic object that denotes a nonempty domain D_α of classes. A sort can denote any nonempty domain specified by a unary predicate symbol. The sorts of $\mathcal{L}$ in a STMM theory $\mathcal{T} = (\mathcal{L}, \Gamma)$ form a partial order $\ll$ that extends the partial order $\preceq_\xi$. If $\mathcal{T} \models \alpha \ll \beta$, then $D_\alpha \subseteq D_\beta$ is true in every model for $\mathcal{T}$. In every theory, $\mathbf{C}$, the sort of classes, is the maximum sort: $\alpha \ll \mathbf{C}$ for every sort α. A compound sort $\alpha \rightharpoonup \beta$ denotes the domain of all functions from the sets in D_α to the sets in D_β. Hence, $\mathbf{V} \rightharpoonup \mathbf{V}$ and $\mathbf{C} \rightharpoonup \mathbf{C}$ both denote the entire domain of functions. Since functions may be partial, $\alpha \rightharpoonup \beta \ll \alpha' \rightharpoonup \beta'$ whenever $\alpha \ll \alpha'$ and $\beta \ll \beta'$.

Sorts are convenient for classifying terms by value. If a term t is defined in a sort α (i.e., $(t \downarrow \alpha)$ holds), then t denotes a class in D_α. A term may be defined in several sorts. For example, if $(t \downarrow \alpha)$ and $\alpha \ll \beta$, then $(t \downarrow \beta)$. Like types, sorts are assigned to terms on the basis of their syntax, but unlike types, sorts may be assigned to nondenoting terms (i.e., terms which are not "type correct"). That is, every term t of $\mathcal{L}$ is assigned a sort $\bar{\sigma}(t)$ whether or not t is defined. A term t is defined in $\bar{\sigma}(t)$ *provided* t *is defined.*

5.4. Function Application

A *function application* of $\mathcal{L}$ is a term of the form $f[a]$ where f and a are terms of $\mathcal{L}$. A function application $f[a]$ is intended to denote the result of applying a function denoted by f to an argument denoted by a. Regardless of what values f and a have, $f[a]$ is a legitimate, well-formed term. However, $f[a]$ is defined only if f denotes a function and a denotes a set in the domain of the denotation of f. If $\alpha = \bar{\sigma}(f)$ is a function sort, $\beta = \bar{\sigma}(f[a])$ is the range sort of α; otherwise $\beta = \mathbf{V}$, the sort of sets.

Function applications provide STMM with the means to express statements involving functions in a very concise manner. For example, the sentence

$$\forall\, x, y : \mathbf{R} \,.\, \sqrt{x} = y \supset x = y^2,$$

where $\mathbf{R}$ denotes the set of real numbers, says that the square of a square root of a number is the number itself. There is no need to mention that the domain of the square function is the set of nonnegative real numbers since $\sqrt{x} = y$ is true only if $\sqrt{x}$ is defined. In traditional set theories like ZF and NBG, the application of a set representing a function can only be expressed in a verbose, indirect way with the use of quantifiers.

5.5. Function Abstraction

A *function abstraction* of $\mathcal{L}$ is a term of the form $(\lambda\, x : \alpha \,.\, t)$. When $(\lambda\, x : \alpha \,.\, t)$ is defined, it denotes a function in the denotation of its assigned sort $\alpha \rightharpoonup \bar{\sigma}(t)$. Together, function applications and function abstractions constitute a convenient lambda-notation for applying and defining classes which denote functions.

5.6. Definite Description

A definite description of $\mathcal{L}$ is a term of the form $(\mathrm{I}\, x : \alpha \,.\, \varphi)$. Since a definite description $(\mathrm{I}\, x : \alpha \,.\, \varphi)$ is defined only if there is a unique x satisfying φ, definite descriptions are exceedingly useful for specifying functions, especially partial functions. For example,

$$\lambda\, x : \mathbf{R} \,.\, \mathrm{I}\, y : \mathbf{R} \,.\, 0 \leq y \wedge y * y = x,$$

where $\mathbf{R}$ denotes the set of real numbers, specifies the standard square root function over the real numbers.

6. Implementation

For the most part, STMM can be implemented like other logics having quantifiers and lambda-notation. However, some of the less common aspects of STMM—undefined terms, definite descriptions, and sorts—require special implementation techniques.

6.1. Undefined Terms and Sorts

The undefined terms and sorts of STMM can be implemented in the same way the undefined terms and sorts of LUTINS are implemented in IMPS.

Reasoning in STMM (like LUTINS) involves proving many obligations of either the form $t\downarrow$ or $(t \downarrow \alpha)$. (Recall that $t\downarrow$ is an abbreviation for $(t \downarrow \bar{\sigma}(t))$.) For example, before a definite description $(\mathrm{I}\, x : \alpha \;.\; t)$ can be "opened up", one must prove $(\mathrm{I}\, x : \alpha \;.\; t)\downarrow$. Also, an assumption of the form $(\forall\, x : \alpha \;.\; \varphi)$ can be instantiated with a term t only if one can prove $(t \downarrow \alpha)$.

An implementation requires an automated mechanism for quickly determining the truth status of *definedness* goals of the form $t\downarrow$ and *sort definedness* goals of the form $(t \downarrow \alpha)$. The mechanism in IMPS for this is built into the IMPS theory-specific expression simplifier [17, 18, 19]. It is able to discharge the great majority of the definedness and sort definedness obligations that arise during the course of an IMPS proof. A similar simplifier-based mechanism could be incorporated into an implementation of STMM.

6.2. Definite Descriptions

Proving assertions containing definite descriptions can be tricky. Special machinery is needed, for instance, for "opening up" definite descriptions known to be defined, handling definite descriptions whose definedness is not known, and simplifying expressions containing definite descriptions of the form $(\mathrm{I}\, x : \alpha \;.\; x = t)$. Chapter 15 of [18] discusses the definite description machinery implemented in IMPS. This machinery could also be used in an implementation of STMM.

6.3. Little Theories

The *little theories method* is a version of the axiomatic method in which a complex system or body of knowledge is described as a network of axiomatic theories linked by theory interpretations. The paper [16] argues that the little theories approach is highly desirable for mechanized mathematics and describes how IMPS supports it. STMM could be implemented with little theories much like IMPS. One would have to define the notion of an *interpretation* of one STMM theory in another (see [13]).

A simple approach would be to define an interpretation of a theory $\mathcal{T}_1$ in a theory $\mathcal{T}_2$ to be a function that maps the individual constants of $\mathcal{T}_1$ to appropriate terms of $\mathcal{T}_2$ and the atomic sorts of $\mathcal{T}_1$ to appropriate sorts, unary predicates, and terms of $\mathcal{T}_2$. Moreover, an interpretation would be required to leave the kernel theory fixed. Operator and predicate symbols would remain fixed and would be defined only in the kernel theory. Since the kernel theory is a subtheory of every other STMM theory, all STMM theories would have the same basic set-theoretic machinery and, in particular, the same operator and predicate symbols defined in the same way.

6.4. Implementation in IMPS

STMM has been implemented in IMPS by using the IMPS logic LUTINS, a partial version of higher-order logic, in place of PFOL. The sort system for STMM is inherited from the sort system for LUTINS with compound sorts of STMM represented by atomic sorts of LUTINS. As a consequence, the support in IMPS for compound sorts of LUTINS is not extended to compound sorts of STMM. This is a significant shortcoming of the implementation that can be fixed only by modifying IMPS itself.

7. Comparison with Other Implemented Set Theories

As we have shown, STMM is a hybrid system fashioned from the following components:

1. PFOL.
2. NBG.
3. A LUTINS-style sort system.
4. Lambda-notation.

The components of STMM are certainly not novel, but the idea of using them together and the way they are combined are novel.

Unlike STMM, none of the six set theories implemented in the mechanized mathematics systems listed in section 1 satisfies all of the design goals given in section 2. In particular, none of them satisfies **DG3** and only Quaife's version of NBG satisfies **DG2**. ZF, NBG, and related set theories can be modified to minimally satisfy **DG4–6**. However, as we noted in section 2, definite description operators, systems for classifying terms by value, and operators for function application and abstraction are awkward in set theories that do not admit proper classes and undefined terms. By virtue of satisfying both **DG2** and **DG3**, STMM satisfies **DG4–6** more effectively than traditional set theories.

As a foundation for mechanized mathematics, STMM is actually much closer to LUTINS than to any of the six implemented set theories. Although LUTINS is a version of Church's simple type theory, it can be viewed as a set theory since sets can be formalized in simple type theory as characteristic functions (in one way or another). As such, LUTINS satisfies all the design goals except **DG2**. Nevertheless, LUTINS is formally less expressive than STMM since simple type theory is less expressive than ZF. Also, in LUTINS polymorphism is a concern because there is more than one type, while in STMM terms denoting functions are effectively polymorphic operators because they can be applied to all sets. STMM has all the virtues of LUTINS as well as the full power of NBG set theory. Moreover, any mathematical reasoning performed in LUTINS can be directly transferred to STMM.

8. Conclusion

We have presented a version of NBG set theory that is intended to serve as a foundation for mechanized mathematics. It is based on familiar, well-understood ideas and principles from predicate logic and mathematics practice. It has the same expressive power as the ZF and NBG set theories and the same convenient machinery for reasoning about functions as LUTINS, the logic of IMPS. It also has a sort system for classifying terms by value, like LUTINS's sort system, that is not a major departure from traditional set theory. And, finally, it can be implemented in much the same way that LUTINS is implemented in IMPS.

A mechanized mathematics system based on a set theory like STMM would be an asset for mechanized mathematics. Since set theory is familiar to nearly all mathematics practitioners, a system based on set theory would likely be more accessible to ordinary students, scientists, and engineers than systems based on type theory. Also, mathematicians might be more inclined to participate in mechanized mathematics if there were more and better mechanized mathematics systems that communicated using set theory.

Notes

[1] Examples include Maple [8] and Mathematica [42].

[2] Examples include Automath [31], Coq [2], EVES [10], HOL [23], IMPS [20, 17], Isabelle [35], Mizar [38], Nqthm [5], Nuprl [9], Otter [26], and PVS [33].

[3] Tarski-Grothendieck set theory is ZF plus an axiom due to A. Tarski [41] that asserts the existence of *universes*—sets closed under the usual set-theoretic operations.

[4] Pronounced as the word "stem".

Acknowledgements

The author expresses special thanks to Dr. Joshua Guttman for his assistance in the development of STMM.

References

1. P. Aczel. *Non-Well-Founded Sets.* CSLI Publications, Stanford, 1988.
2. B. Barros et al. The Coq proof assistant reference manual, version 6.1. Available at `ftp://ftp.inria.fr/INRIA/coq/V6.1/doc/Reference-Manual.dvi.gz`, 1997.
3. M. Beeson. Formalizing constructive mathematics: Why and how? In F. Richman, editor, *Constructive Mathematics: Proceedings, New Mexico, 1980*, volume 873 of *Lecture Notes in Mathematics*, pages 146–190. Springer-Verlag, 1981.
4. M. J. Beeson. *Foundations of Constructive Mathematics.* Springer-Verlag, Berlin, 1985.
5. R. Boyer and J Moore. *A Computational Logic Handbook.* Academic Press, 1988.
6. T. Burge. *Truth and Some Referential Devices.* PhD thesis, Princeton University, 1971.
7. T. Burge. Truth and singular terms. In K. Lambert, editor, *Philosophical Applications of Free Logic*, pages 189–204. Oxford University Press, 1991.
8. B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Language Reference Manual.* Springer-Verlag, 1991.
9. 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.
10. D. Craigen, S. Kromodimoeljo, I. Meisels, B. Pase, and M. Saaltink. EVES: An overview. Technical Report CP-91-5402-43, ORA Corporation, 1991.
11. W. M. Farmer. A partial functions version of Church's simple theory of types. *Journal of Symbolic Logic*, 55:1269–91, 1990.
12. W. M. Farmer. A simple type theory with partial functions and subtypes. *Annals of Pure and Applied Logic*, 64:211–240, 1993.
13. W. M. Farmer. Theory interpretation in simple type theory. In J. Heering et al., editor, *Higher-Order Algebra, Logic, and Term Rewriting*, volume 816 of *Lecture Notes in Computer Science*, pages 96–123. Springer-Verlag, 1994.
14. W. M. Farmer. Reasoning about partial functions with the aid of a computer. *Erkenntnis*, 43:279–294, 1995.
15. W. M. Farmer and J. D. Guttman. A set theory with support for partial functions. *Studia Logica*, 66:59–78, 2000.
16. W. M. Farmer, J. D. Guttman, and F. J. Thayer. Little theories. In D. Kapur, editor, *Automated Deduction—CADE-11*, volume 607 of *Lecture Notes in Computer Science*, pages 567–581. Springer-Verlag, 1992.
17. W. M. Farmer, J. D. Guttman, and F. J. Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11:213–248, 1993.

18. W. M. Farmer, J. D. Guttman, and F. J. Thayer. The IMPS user's manual. Technical Report M-93B138, The MITRE Corporation, 1993. Available at `http://imps.mcmaster.ca/`.
19. W. M. Farmer, J. D. Guttman, and F. J. Thayer. Contexts in mathematical reasoning and computation. *Journal of Symbolic Computation*, 19:201–216, 1995.
20. W. M. Farmer, J. D. Guttman, and F. J. Thayer Fábrega. IMPS: An updated system description. In M. McRobbie and J. Slaney, editors, *Automated Deduction—CADE-13*, volume 1104 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, 1996.
21. K. Gödel. *The Consistency of the Axiom of Choice and the Generalized Continuum Hypothesis with the Axioms of Set Theory*, volume 3 of *Annals of Mathematical Studies*. Princeton University Press, 1940.
22. M. Gordon. Set theory, higher order logic or both? In J. Grundy and J. Harrison, editors, *Theorem Proving in Higher Order Logic: 9th International Conference, TPHOLs'96*, volume 1125 of *Lecture Notes in Computer Science*, pages 191–202. Springer-Verlag, 1996.
23. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.
24. K. Kunen. *Set Theory: An Introduction to Independence Proofs*. North-Holland, 1980.
25. L. Lamport and L. C. Paulson. Should your specification language be typed? *ACM Transactions on Programming Languages and Systems*, 21:502–526, 1999.
26. W. McCune. OTTER 2.0. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Computer Science*, pages 663–664. Springer-Verlag, 1990.
27. N. D. Megill. *Metamath: A Computer Language for Pure Mathematics*. 1997. Available at `http://www.shore.net/~ndm/java/mm.html`.
28. E. Mendelson. *Introduction to Mathematical Logic*. Van Nostrand, 1964.
29. L. G. Monk. PDLM: A Proof Development Language for Mathematics. Technical Report M86-37, The MITRE Corporation, Bedford, Massachusetts, 1986.
30. R. Montague. Semantic closure and non-finite axiomatizability. In *Infinitistic Methods*, pages 45–69. Pergamon, 1961.
31. 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.
32. I. L. Novak. A construction for models of consistent systems. *Fundamenta Mathematicae*, 37:87–110, 1950.
33. 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.
34. L. C. Paulson. Set theory for verification: I. From foundations to functions. *Journal of Automated Reasoning*, 11:353–389, 1993.
35. L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
36. A. Quaife. Automated deduction in von Neumann-Bernays-Gödel set theory. *Journal of Automated Reasoning*, 8:91–147, 1993.
37. J. B. Rosser and H. Wang. Non-standard models for formal logics. *Journal of Symbolic Logic*, 15:113–129, 1950.

38. P. Rudnicki. An overview of the MIZAR project. Technical report, Department of Computing Science, University of Alberta, 1992.
39. R. Schock. *Logics without Existence Assumptions.* Almqvist & Wiksells, Stockholm, Sweden, 1968.
40. J. Shoenfield. A relative consistency proof. *Journal of Symbolic Logic*, 19:21–28, 1954.
41. A. Tarski. Über unerreichbare Kardinalzahlen. *Fundamenta Mathematicae*, 30:68–89, 1938.
42. S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, 1991.