

# Formalizing Undefinedness Arising in Calculus\*

William M. Farmer

McMaster University  
Hamilton, Ontario, Canada  
wmfarmer@mcmaster.ca

February 20, 2005

**Abstract.** Undefined terms are commonplace in mathematics, particularly in calculus. The *traditional approach to undefinedness* in mathematical practice is to treat undefined terms as legitimate, nondenoting terms that can be components of meaningful statements. The traditional approach enables statements about partial functions and undefined terms to be stated very concisely. Unfortunately, the traditional approach cannot be easily employed in a standard logic in which all functions are total and all terms are defined, but it can be directly formalized in a standard logic if the logic is modified slightly to admit undefined terms and statements about definedness. This paper demonstrates this by defining a version of simple type theory called Simple Type Theory with Undefinedness (STTWU) and then formalizing in STTWU examples of undefinedness arising in calculus. The examples are taken from M. Spivak's well-known textbook *Calculus*.

## 1 Introduction

A mathematical term is *undefined* if it has no prescribed meaning or if it denotes a value that does not exist. Undefined terms are commonplace in mathematics, particularly in calculus. As a result, any approach to formalizing mathematics must include a method for handling undefinedness.

There are two principal sources of undefinedness. The first source are terms that denote an application of a function. A function  $f$  usually has both a *domain of definition*  $D_f$  consisting of the values at which it is defined and a *domain of application*  $D_f^*$  consisting of the values to which it may be applied. (The domain of definition of a function is usually called simply the *domain* of the function.) These two domains are not always the same. A *function application* is a term  $f(a)$  that denotes the application of a function  $f$  to an argument  $a \in D_f^*$ .  $f(a)$  is *undefined* if  $a \notin D_f$ . We will say that a function is *partial* if  $D_f \neq D_f^*$  and *total* if  $D_f = D_f^*$ .

As an example, consider the square root function  $\sqrt{\phantom{x}} : \mathbf{R} \rightarrow \mathbf{R}$ , where  $\mathbf{R}$  denotes the set of real numbers.  $\sqrt{\phantom{x}}$  is a partial function; it is defined only on

---

\* © Springer-Verlag. Published in D. Basin and M. Rusinowitch, eds., *Automated Reasoning, LNCS*, 3097:475-489, 2004.

the nonnegative real numbers, but it can be applied to negative real numbers as well. That is,  $D_{\sqrt{\cdot}} = \{x \in \mathbf{R} \mid 0 \leq x\}$  and  $D_{\sqrt{\cdot}^*} = \mathbf{R}$ . Hence, a statement like

$$\forall x : \mathbf{R} . 0 \leq x \Rightarrow (\sqrt{x})^2 = x$$

makes perfectly good sense even though  $\sqrt{x}$  is undefined when  $x < 0$ .

The second source of undefinedness are terms that are intended to uniquely describe a value. A *definite description* is a term  $t$  of the form “the  $x$  that has the property  $P$ ”.  $t$  is *undefined* if there is no unique  $x$  (i.e., none or more than one) that has property  $P$ . Definite descriptions are quite common in mathematics but often occur in a disguised form. For example, “the limit of  $\sin \frac{1}{x}$  as  $x$  approaches 0” is a definite description—which is undefined since the limit does not exist.

There is a *traditional approach to undefinedness* that is widely practiced in mathematics and even taught to some extent to students in high school. This approach treats undefined terms as legitimate, nondenoting terms that can be components of meaningful statements. The traditional approach is based on three principles:

1. Atomic terms (i.e., variables and constants) are always defined—they always denote something.
2. Compound terms may be undefined. A function application  $f(a)$  is undefined if  $f$  is undefined,  $a$  is undefined, or  $a \notin D_f$ . A definite description “the  $x$  that has property  $P$ ” is undefined if there is no  $x$  that has property  $P$  or there is more than one  $x$  that has property  $P$ .
3. Formulas are always true or false, and hence, are always defined. To ensure the definedness of formulas, a function application  $p(a)$  formed by applying a predicate  $p$  to an argument  $a$  is *false* if  $p$  is undefined,  $a$  is undefined, or  $a \notin D_p$ .

Formalizing the traditional approach to undefinedness is problematic. The traditional approach works smoothly in informal mathematics, but it cannot be easily employed in mathematics formalized in standard logics like first-order logic or simple type theory. This is due to the fact that in a standard logic all functions are total and all terms denote some value. As a result, in a standard logic partial functions must be represented by total functions and undefined terms must be given a value.

The mathematics formalizer has basically three choices for how to formalize undefinedness. The first choice is to formalize partial functions and undefined terms in a standard logic. There are various methods by which this can be done (see [8, 21]). Each of these methods, however, has the disadvantage that it is a significant departure from the traditional approach. This means in practice that a concise informal mathematical statement  $S$  involving partial functions or undefinedness is often represented by a verbose formal mathematical statement  $S'$  in which unstated, but implicit, definedness assumptions within  $S$  are explicitly represented.

The second choice is to select or develop a three-valued logic in which the traditional approach can be formalized. Such a logic would admit both undefined terms and undefined formulas. The logic needs to be three valued since

formulas can be undefined as well as true and false. For example, M. Kerber and M. Kohlhase propose in [19] a three-valued version of many-sorted first-order logic that is intended to support the traditional approach to undefinedness. A three-valued logic of this kind provides a very flexible basis for formalizing and reasoning about undefinedness. However, it is nevertheless a significant departure from the traditional approach: it is very unusual in mathematical practice to use truth values beyond true and false. Moreover, the presence of a third truth value makes the logic more complicated to use and implement.

It is possible to *directly* formalize the traditional approach in a standard logic if the logic is modified slightly to admit undefined terms and statements about definedness but not undefined formulas (see [11]). The resulting logic remains two-valued and can be viewed as a more convenient version of the original standard logic. Let us call a logic of this kind a *logic with undefinedness*. As far as we know, the first example of a logic with undefinedness is a version of first-order logic presented by R. Schock in [23]. Other logics with undefinedness have been derived from first-order logic [3–6, 15, 18, 20, 22], simple type theory [8–10], and set theory [12, 15].

The third choice is to select or develop an appropriate logic with undefinedness. Using a logic with undefinedness to formalize the traditional approach has two advantages and one disadvantage. The first advantage is that the use of the traditional approach in informal mathematics can be preserved in the formalized mathematics. This helps keep the formalized mathematics close to the informal mathematics in form and meaning. The second advantage is that assumptions about the definedness of terms and functions often do not have to be made explicit. This helps keep the formalized mathematics concise. The disadvantage, of course, is that one is committed to using a nonstandard logic that is based on (slightly) different principles and requires different techniques for implementation.

This disadvantage is not as bad as one might think. By virtue of directly formalizing the traditional method, a logic with undefinedness is closer to traditional mathematical practice than the corresponding standard logic in which all functions are total and all terms are defined. Hence, logics with undefinedness are more practice-oriented than standard logics. Moreover, the logic LUTINS [10], a logic with undefinedness derived from simple type theory, is the basis for the IMPS theorem proving system [16, 17]. IMPS has been used to prove hundreds of theorems in traditional mathematics, especially in mathematical analysis. Most of these theorems involve undefinedness in some manner. The techniques used to implement LUTINS in IMPS have been thoroughly tested and can be applied to the implementation of other logics with undefinedness.

Even though LUTINS has been successfully implemented in IMPS and has proven to be highly effective for mechanizing traditional mathematics, there is still a great deal of scepticism and misunderstanding concerning logics with undefinedness. The goal of this paper is to try to dispel some of this scepticism and misunderstanding by illustrating how undefinedness arising in calculus can be directly formalized in a very simple higher-order logic called Simple Type

Theory with Undefinedness (STTWU). We will show that the conciseness that comes from the use of the traditional approach can be fully preserved in a logic like STTWU. For example,

$$f(x) = \sqrt{x^2 - 1},$$

a common-style definition of a partial function in which the domain of the function is implicitly defined, can be formalized precisely in STTWU as

$$\forall x . f(x) \simeq \sqrt{x^2 - 1}$$

(where  $a \simeq b$  means  $a$  and  $b$  are both defined and equal or both undefined).

All of our examples from calculus come from M. Spivak’s well-known textbook *Calculus* [24]. Spivak’s book is a masterpiece; it is elegantly rigorous, replete with interesting examples and exercises, and exceptionally careful about important issues such as undefinedness that are not always given proper attention in other calculus textbooks. More than just a book on calculus, *Calculus* is also an uncompromising introduction to mathematical practice and mathematical thinking. It is an excellent place to see how undefinedness is handled in standard mathematical practice.

The paper is organized as follows. Section 2 presents the syntax and semantics of STTWU, a version of simple type theory that directly formalizes the traditional approach to undefinedness. Section 2 also give a proof system for STTWU. Section 3 describes a theory of STTWU that formalizes the 13 properties of the real numbers on which Spivak’s *Calculus* is based. How partial functions are defined in *Calculus* and how their definitions are formalized in STTWU is the subject of section 4. Section 5 deals with the important notion in calculus of a limit of a function at a point, which is a rich source of undefinedness. Finally, a conclusion and a recommendation are given in section 6.

## 2 Simple Type Theory with Undefinedness

In this section we present a version of simple type theory called *Simple Type Theory with Undefinedness* (STTWU) that formalizes the traditional approach to undefinedness. STTWU is a variant of Church’s type theory [7] with a standard syntax but a nonstandard semantics. STTWU is very similar to **PF** [8], **PF\*** [9], and LUTINS [10], the logic of the IMPS. **PF** and **PF\*** are simple versions of LUTINS that are primarily intended for study unlike LUTINS. STTWU is much simpler than these three logics but much less practical than **PF\*** and LUTINS. The definition of STTWU will show that the traditional approach can be formalized in Church’s type theory by just a small modification of its semantics.

### 2.1 Syntax

The syntax of STTWU is exactly the same syntax as the syntax of STT, a very simple variant of Church’s type system presented in [13]. STTWU has two kinds of

syntactic objects. “Expressions” denote values including the truth values T (true) and F (false); they play the role of both terms and formulas. “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.

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

$$\begin{aligned} \mathbf{T1} \quad & \frac{}{\mathbf{type}[\iota]} \quad (\mathbf{Type\ of\ individuals}) \\ \mathbf{T2} \quad & \frac{}{\mathbf{type}[*]} \quad (\mathbf{Type\ of\ truth\ values}) \\ \mathbf{T3} \quad & \frac{\mathbf{type}[\alpha], \mathbf{type}[\beta]}{\mathbf{type}[(\alpha \rightarrow \beta)]} \quad (\mathbf{Function\ type}) \end{aligned}$$

Let  $\mathcal{T}$  denote the set of types of STTWU.

The *logical symbols* of STTWU are:

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

A *language* of STTWU is a pair  $L = (\mathcal{C}, \tau)$  where  $\mathcal{C}$  is a set of symbols called *constants* and  $\tau : \mathcal{C} \rightarrow \mathcal{T}$  is a total function. That is, a language is a set of symbols with assigned types (what is also called a “signature”).

An *expression*  $E$  of *type*  $\alpha$  of an STTWU 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  $\alpha$  of  $L$ .

$$\begin{aligned} \mathbf{E1} \quad & \frac{x \in \mathcal{V}, \mathbf{type}[\alpha]}{\mathbf{expr}_L[(x : \alpha), \alpha]} \quad (\mathbf{Variable}) \\ \mathbf{E2} \quad & \frac{c \in \mathcal{C}}{\mathbf{expr}_L[c, \tau(c)]} \quad (\mathbf{Constant}) \\ \mathbf{E3} \quad & \frac{\mathbf{expr}_L[A, \alpha], \mathbf{expr}_L[F, (\alpha \rightarrow \beta)]}{\mathbf{expr}_L[(F @ A), \beta]} \quad (\mathbf{Function\ application}) \\ \mathbf{E4} \quad & \frac{x \in \mathcal{V}, \mathbf{type}[\alpha], \mathbf{expr}_L[B, \beta]}{\mathbf{expr}_L[(\lambda x : \alpha . B), (\alpha \rightarrow \beta)]} \quad (\mathbf{Function\ abstraction}) \\ \mathbf{E5} \quad & \frac{\mathbf{expr}_L[E_1, \alpha], \mathbf{expr}_L[E_2, \alpha]}{\mathbf{expr}_L[(E_1 = E_2), *]} \quad (\mathbf{Equality}) \\ \mathbf{E6} \quad & \frac{x \in \mathcal{V}, \mathbf{type}[\alpha], \mathbf{expr}_L[A, *]}{\mathbf{expr}_L[(I x : \alpha . A), \alpha]} \quad (\mathbf{Definite\ description}) \end{aligned}$$

We will see shortly that the value of a definite description  $(Ix : \alpha . A)$  is the unique value  $x$  of type  $\alpha$  satisfying  $A$  if it exists and is “undefined” otherwise.

“Free variable”, “closed expression”, and similar notions are defined in the obvious way. An expression of  $L$  is a *formula* if it is of type  $*$ , a *sentence* if it is a closed formula, and a *predicate* if it is of type  $(\alpha \rightarrow *)$  for any  $\alpha \in \mathcal{T}$ . Let  $A_\alpha, B_\alpha, C_\alpha, \dots$  denote expressions of type  $\alpha$ . 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  $F(A)$ .

## 2.2 Semantics

The semantics of STTWU is the same as the semantics of STT except that:

1. A model contains partial and total functions instead of just total functions.
2. The value of an “undefined” function application is F if it is a formula and is undefined if it is not a formula.
3. The value of a function abstraction is a function that is possibly partial.
4. The value of an equality is F if the value of either of its arguments is undefined.
5. The value of an “undefined” definite description is F if it is a formula and is undefined if it is not a formula.

A *model*<sup>1</sup> for a language  $L = (\mathcal{C}, \tau)$  of STTWU is a pair  $M = (\mathcal{D}, I)$  where:

1.  $\mathcal{D} = \{D_\alpha : \alpha \in \mathcal{T}\}$  is a set of nonempty domains (sets).
2.  $D_* = \{\mathbf{T}, \mathbf{F}\}$ , the domain of truth values.
3. For  $\alpha, \beta \in \mathcal{T}$ ,  $D_{\alpha \rightarrow \beta}$  is the set of all *total* functions from  $D_\alpha$  to  $D_\beta$  if  $\beta = *$  and is the set of all *partial and total* functions from  $D_\alpha$  to  $D_\beta$  if  $\beta \neq *$ .<sup>2</sup>
4.  $I$  maps each  $c \in \mathcal{C}$  to a member of  $D_{\tau(c)}$ .

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

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

<sup>1</sup> This is the definition of a *standard model* for STTWU. There is also the notion of a *general model* for STTWU in which functions domains are not fully “inhabited” (see [14]).

<sup>2</sup> The condition that a domain  $D_{\alpha \rightarrow *}$  contains only total functions is needed to ensure that the law of extensionality holds for predicates. This condition is weaker than the condition used in the semantics for LUTINS and its simple versions, **PF** and **PF\***. In these logics, a domain  $D_\gamma$  contains only total functions iff  $\gamma$  has the form  $(\alpha_1 \rightarrow (\alpha_2 \rightarrow \dots (\alpha_n \rightarrow *) \dots))$  where  $n \geq 1$ . The weaker condition, which is due to A. Stump [25], yields a semantics that is somewhat simpler.

1. Let  $E$  be a variable (i.e.,  $E$  is of the form  $(x : \alpha)$ ). Then  $V_\varphi^M(E) = \varphi(E)$ .
2. Let  $E$  be a constant of  $L$  (i.e.,  $E \in \mathcal{C}$ ). Then  $V_\varphi^M(E) = I(E)$ .
3. Let  $E_\beta$  be of the form  $(F @ A)$ . If  $V_\varphi^M(F)$  is defined,  $V_\varphi^M(A)$  is defined, and  $V_\varphi^M(A)$  is in the domain of  $V_\varphi^M(F)$ , then  $V_\varphi^M(E_\alpha) = V_\varphi^M(F)(V_\varphi^M(A))$ , the result of applying the function  $V_\varphi^M(F)$  to the argument  $V_\varphi^M(A)$ . Otherwise,  $V_\varphi^M(E_\beta) = \mathbb{F}$  if  $\beta = *$  and  $V_\varphi^M(E_\beta)$  is undefined if  $\beta \neq *$ .
4. Let  $E_{\alpha \rightarrow \beta}$  be of the form  $(\lambda x : \alpha . B)$ . Then  $V_\varphi^M(E_{\alpha \rightarrow \beta})$  is the (partial or total) function  $f : D_\alpha \rightarrow D_\beta$  such that, for each  $d \in D_\alpha$ ,  $f(d) = V_{\varphi[(x:\alpha) \mapsto d]}^M(B)$  if  $V_{\varphi[(x:\alpha) \mapsto d]}^M(B)$  is defined and  $f(d)$  is undefined if  $V_{\varphi[(x:\alpha) \mapsto d]}^M(B)$  is undefined.
5. Let  $E_*$  be of the form  $(E_1 = E_2)$ . If  $V_\varphi^M(E_1)$  is defined,  $V_\varphi^M(E_2)$  is defined, and  $V_\varphi^M(E_1) = V_\varphi^M(E_2)$ , then  $V_\varphi^M(E_*) = \mathbb{T}$ ; otherwise  $V_\varphi^M(E_*) = \mathbb{F}$ .
6. Let  $E_\alpha$  be of the form  $(\text{I } x : \alpha . A)$ . If there is a unique  $d \in D_\alpha$  such that  $V_{\varphi[(x:\alpha) \mapsto d]}^M(A) = \mathbb{T}$ , then  $V_\varphi^M(E_\alpha) = d$ . Otherwise,  $V_\varphi^M(E_\alpha) = \mathbb{F}$  if  $\alpha = *$  and  $V_\varphi^M(E_\alpha)$  is undefined if  $\alpha \neq *$ .

Let  $E$  be an expression of type  $\alpha$  of  $L$ . When  $V_\varphi^M(E)$  is defined,  $V_\varphi^M(E)$  is called the *value* of  $E$  in  $M$  with respect to  $\varphi$  and  $V_\varphi^M(E) \in D_\alpha$ . Whenever  $E$  is a formula,  $V_\varphi^M(E)$  is defined. A formula  $A$  is *valid* in  $M$ , written  $M \models A$ , if  $V_\varphi^M(A) = \mathbb{T}$  for all variable assignments  $\varphi$  into  $M$ . A *theory* of STTWU is a pair  $T = (L, \Gamma)$  where  $L$  is a language of STTWU and  $\Gamma$  is a set of sentences of  $L$  called the *axioms* of  $T$ . A *model* of  $T$  is a model  $M$  for  $L$  such that  $M \models B$  for all  $B \in \Gamma$ .

### 2.3 Definitions and Abbreviations

We define in STTWU the standard propositional connectives and quantifiers as well as some special operators concerning definedness.

$\mathbb{T}$	means	$(\lambda x : * . x) = (\lambda x : * . x)$ .
$\mathbb{F}$	means	$(\lambda x : * . \mathbb{T}) = (\lambda x : * . x)$ .
$\neg A_*$	means	$A_* = \mathbb{F}$ .
$(A_\alpha \neq B_\alpha)$	means	$\neg(A_\alpha = B_\alpha)$ .
$(A_* \wedge B_*)$	means	$(\lambda f : (* \rightarrow (* \rightarrow *)) . f(\mathbb{T})(\mathbb{T})) =$ $(\lambda f : (* \rightarrow (* \rightarrow *)) . f(A_*)(B_*))$ .
$(A_* \vee B_*)$	means	$\neg(\neg A_* \wedge \neg B_*)$ .
$(A_* \Rightarrow B_*)$	means	$\neg A_* \vee B_*$ .
$(A_* \Leftrightarrow B_*)$	means	$A_* = B_*$ .
$(\forall x : \alpha . A_*)$	means	$(\lambda x : \alpha . A_*) = (\lambda x : \alpha . \mathbb{T})$ .
$(\exists x : \alpha . A_*)$	means	$\neg(\forall x : \alpha . \neg A_*)$ .
$(A_\alpha \downarrow)$	means	$\exists x : \alpha . x = A_\alpha$ where $(x : \alpha)$ does not occur in $A_\alpha$ .
$(A_\alpha \uparrow)$	means	$\neg(A_\alpha \downarrow)$ .
$(A_\alpha \simeq B_\alpha)$	means	$(A_\alpha \downarrow \vee B_\alpha \downarrow) \Rightarrow A_\alpha = B_\alpha$ .

$$\begin{array}{ll} \perp_\alpha & \text{means } \text{I } x : \alpha . x \neq x. \\ \text{if}(A_*, B_\alpha, C_\alpha) & \text{means } \text{I } x : \alpha . (A_* \Rightarrow x = B_\alpha) \wedge (\neg A_* \Rightarrow x = C_\alpha) \\ & \text{where } (x : \alpha) \text{ does not occur in } A_*, B_\alpha, \text{ or } C_\alpha. \end{array}$$

Notice that we are using the syntactic conventions mentioned in section 2.1. For example, the meaning of  $\mathbf{T}$  is officially the expression

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

$(A_\alpha \downarrow)$  says that  $A_\alpha$  is defined,  $(A_\alpha \uparrow)$  says that  $A_\alpha$  is undefined, and  $A_\alpha \simeq B_\alpha$  says that  $A_\alpha$  and  $B_\alpha$  are *quasi-equal*, i.e., that  $A_\alpha$  and  $B_\alpha$  are either both defined and equal or both undefined.  $\perp_\alpha$  is a canonical undefined expression of type  $\alpha$ .  $\text{if}$  is an if-then-else expression constructor such that  $\text{if}(A_*, B_\alpha, C_\alpha)$  denotes  $B_\alpha$  if  $A_*$  holds and denotes  $C_\alpha$  if  $\neg A_*$  holds.

We will write a formula of the form  $\square x_1 : \alpha . \dots \square x_n : \alpha . A$  as simply  $\square x_1, \dots, x_n : \alpha . A$  where  $\square$  is  $\forall$  or  $\exists$ . If we fix the type of a variable  $x$ , say to  $\alpha$ , then an expression of the form  $\square x : \alpha . E$  may be written as simply  $\square x . E$  where  $\square$  is  $\lambda, \text{I}, \forall$ , or  $\exists$ . If desired, all types can be removed from an expression by fixing the types of the variables occurring in the expression.

## 2.4 Proof System

We present now a Hilbert-style proof system for  $\text{STTwU}$  called  $\mathbf{A}_u$  that is sound and complete with respect to the general models semantics for  $\text{STTwU}$  [14]. It is a modification of the proof system  $\mathbf{A}$  for  $\text{STT}$  given in [13] which is based on P. Andrews' proof system [1, 2] for Church's type theory.

Define  $B_\beta[(x : \alpha) \mapsto A_\alpha]$  to be the result of simultaneously replacing each free occurrence of  $(x : \alpha)$  in  $B_\beta$  by an occurrence of  $A_\alpha$ . Let  $(\exists! x : \alpha . A)$  mean

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

where  $(y : \alpha)$  does not occur in  $A$ . This formula asserts there exists a unique value  $x$  of type  $\alpha$  that satisfies  $A$ .

For a language  $L = (\mathcal{C}, \tau)$ , the proof system  $\mathbf{A}_u$  consists of the following sixteen axiom schemas and two rules of inference:

### A1 (Truth Values)

$$\forall f : (* \rightarrow *) . (f(\mathbf{T}) \wedge f(\mathbf{F})) \Leftrightarrow (\forall x : * . f(x)).$$

### A2 (Leibniz' Law)

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

### A3 (Extensionality)

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

**A4 (Beta-Reduction)**

$$A_\alpha \downarrow \Rightarrow (\lambda x : \alpha . B_\beta)(A_\alpha) \simeq B_\beta[(x : \alpha) \mapsto A_\alpha]$$

provided  $A_\alpha$  is free for  $(x : \alpha)$  in  $B_\beta$ .

**A5 (Equality and Quasi-Quality)**

$$A_\alpha \downarrow \Rightarrow (B_\alpha \downarrow \Rightarrow (A_\alpha \simeq B_\alpha) \simeq (A_\alpha = B_\alpha)).$$

**A6 (Expressions of Type  $*$  are Defined)**

$$A_* \downarrow .$$

**A7 (Variables are Defined)**

$$(x : \alpha) \downarrow \quad \text{where } x \in \mathcal{V} \text{ and } \alpha \in \mathcal{T}.$$

**A8 (Constants are Defined)**

$$c \downarrow \quad \text{where } c \in \mathcal{C}.$$

**A9 (Function Abstractions are Defined)**

$$(\lambda x : \alpha . B_\beta) \downarrow$$

**A10 (Improper Function Application)**

$$(F_{\alpha \rightarrow \beta} \uparrow \vee A_\alpha \uparrow) \Rightarrow F_{\alpha \rightarrow \beta}(A_\alpha) \uparrow \quad \text{where } \beta \neq *.$$

**A11 (Improper Predicate Application)**

$$(F_{\alpha \rightarrow *} \uparrow \vee A_\alpha \uparrow) \Rightarrow \neg F_{\alpha \rightarrow *}(A_\alpha).$$

**A12 (Improper Equality)**

$$(A_\alpha \uparrow \vee B_\alpha \uparrow) \Rightarrow \neg(A_\alpha = B_\alpha).$$

**A13 (Proper Definite Description of Type  $\alpha \neq *$ )**

$$(\exists ! x : \alpha . A_*) \Rightarrow ((Ix : \alpha . A_*) \downarrow \wedge A_*[(x : \alpha) \mapsto (Ix : \alpha . A_*)])$$

where  $\alpha \neq *$  and provided  $(Ix : \alpha . A_*)$  is free for  $(x : \alpha)$  in  $A_*$ .

**A14 (Improper Definite Description of Type  $\alpha \neq *$ )**

$$\neg(\exists ! x : \alpha . A_*) \Rightarrow (Ix : \alpha . A_*) \uparrow \quad \text{where } \alpha \neq *.$$

**A15 (Proper Definite Description of Type  $*$ )**

$$(\exists ! x : * . A_*) \Rightarrow A_*[(x : *) \mapsto (Ix : * . A_*)]$$

provided  $(Ix : * . A_*)$  is free for  $(x : *)$  in  $A_*$ .

**A16 (Improper Definite Description of Type  $*$ )**

$$\neg(\exists ! x : * . A_*) \Rightarrow \neg(Ix : * . A_*).$$

**R1 (Modus Ponens)** From  $A_*$  and  $A_* \Rightarrow B_*$  infer  $B_*$ .

**R2 (Quasi-Equality Substitution)** From  $A_\alpha \simeq B_\alpha$  and  $C_*$  infer the result of replacing one occurrence of  $A_\alpha$  in  $C_*$  by an occurrence of  $B_\alpha$ , provided that the occurrence of  $A_\alpha$  in  $C_*$  is not immediately preceded by  $\lambda$ .

$\mathbf{A}_u$  is sound and complete with respect to the general models semantics for  $\text{STTWU}$  [14]. The completeness proof is closely related to the completeness proofs for  $\mathbf{PF}$  [8] and  $\mathbf{PF}^*$  [9]. All the standard laws of predicate logic hold in  $\text{STTWU}$  except some of those involving equality and instantiation. However, the laws of equality and instantiation do hold if certain “definedness requirements” are satisfied. See [8, 9, 14] for details.

### 3 A Theory of the Real Numbers

Spivak’s development of calculus in his textbook *Calculus* [24] begins with a presentation of 13 basic properties of the real numbers [24, pp. 9, 113]. These properties are essentially the axioms of the theory of a complete ordered field—which has exactly one model up to isomorphism, namely, the standard model of the real numbers.

We will begin our exploration of undefinedness in Spivak’s *Calculus* by formulating the 13 properties as a theory in  $\text{STTWU}$ . Let  $\text{COF} = (L, \Gamma)$  be the theory of  $\text{STTWU}$  such that:

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

Constant $c$	Type $\tau(c)$
$0, 1$	$\iota$
$-, ^{-1}$	$\iota \rightarrow \iota$
<b>pos</b>	$\iota \rightarrow *$
$+, \cdot$	$\iota \rightarrow (\iota \rightarrow \iota)$
$<, \leq$	$\iota \rightarrow (\iota \rightarrow *)$
<b>ub, lub</b>	$(\iota \rightarrow *) \rightarrow (\iota \rightarrow *)$

Note: The type  $\iota$  is being used to represent the set of real numbers.

- $\Gamma$  is the set of the 19 formulas given below. We assume that the variables  $a, b, c$  are of type  $\iota$  and the variable  $s$  is of type  $(\iota \rightarrow *)$ .

**P1**  $\forall a, b, c. a + (b + c) = (a + b) + c.$

**P2**  $\forall a. a + 0 = a \wedge 0 + a = a.$

**P3**  $\forall a. a + -a = 0 \wedge -a + a = 0.$

**P4**  $\forall a, b. a + b = b + a.$

**P5**  $\forall a, b, c. a \cdot (b \cdot c) = (a \cdot b) \cdot c.$

**P6a**  $\forall a. a \cdot 1 = a \wedge 1 \cdot a = a.$

**P6b**  $0 \neq 1.$

- P7a**  $\forall a . a \neq 0 \Rightarrow (a \cdot a^{-1} = 1 \wedge a^{-1} \cdot a = 1).$   
**P7b**  $0^{-1} \uparrow.$   
**P8**  $\forall a, b . a \cdot b = b \cdot a.$   
**P9**  $\forall a, b, c . a \cdot (b + c) = (a \cdot b) + (a \cdot c).$   
**P10**  $\forall a . (a = 0 \wedge \neg \text{pos}(a) \wedge \neg \text{pos}(-a)) \vee$   
 $(a \neq 0 \wedge \text{pos}(a) \wedge \neg \text{pos}(-a)) \vee$   
 $(a \neq 0 \wedge \neg \text{pos}(a) \wedge \text{pos}(-a)).$   
**P11**  $\forall a, b . (\text{pos}(a) \wedge \text{pos}(b)) \Rightarrow \text{pos}(a + b).$   
**P12**  $\forall a, b . (\text{pos}(a) \wedge \text{pos}(b)) \Rightarrow \text{pos}(a \cdot b).$   
**D1**  $\forall a, b . a < b \Leftrightarrow \text{pos}(b - a).$   
**D2**  $\forall a, b . a \leq b \Leftrightarrow (a < b \vee a = b).$   
**D3**  $\forall s, a . \text{ub}(s)(a) \Leftrightarrow (\forall b . s(b) \Rightarrow b \leq a).$   
**D4**  $\forall s, a . \text{lub}(s)(a) \Leftrightarrow (\text{ub}(s)(a) \wedge (\forall b . \text{ub}(s)(b) \Rightarrow a \leq b)).$   
**P13**  $\forall s . ((\exists a . s(a)) \wedge (\exists a . \text{ub}(s)(a)) \Rightarrow \exists a . \text{lub}(s)(a)).$

Notes:

1. Axioms **P1–P13** correspond to Spivak’s 13 properties P1–P13. Properties P6 and P7 are both expressed by pairs of axioms. Axioms **D1–D4** are definitions.
2. We write the additive and multiplicative inverses of  $a$  as  $-a$  and  $a^{-1}$  instead of as  $-(a)$  and  $^{-1}(a)$ , respectively.
3.  $+$  and  $*$  are formalized by constants of type  $(\iota \rightarrow (\iota \rightarrow \iota))$  representing curried functions. However, we write the application of  $+$  and  $*$  using infix notation, e.g., we write  $a + b$  instead of  $+(a)(b)$ .  $<$  and  $\leq$  are handled in a similar way. In our examples below, we will also write  $a - b$  instead of  $a + -b$  and  $\frac{a}{b}$  instead of  $a \cdot b^{-1}$ .
4.  $\text{pos}$  is a predicate that represents the set of positive real numbers.
5.  $\text{ub}(s)(a)$  and  $\text{lub}(s)(a)$  say that  $a$  is an upper bound of  $s$  and  $a$  is the least upper bound of  $s$ , respectively.
6. Axiom **P13** expresses the *completeness principle* of the real numbers, i.e., that every nonempty set of real numbers that has an upper bound has a least upper bound.

COF is an extremely direct formalization of Spivak’s 13 properties. (The reader is invited to check this for herself.) The only significant departure is Axiom **P7b**. The undefinedness of the multiplicative inverse of 0 is not stated in property P7. However, Spivak says in the text that “ $0^{-1}$  is meaningless” and implicitly that  $0^{-1}$  is always undefined [24, p. 6].

COF is categorical, i.e., it has exactly one model  $(\mathcal{D}, I)$  up to isomorphism where  $D_\iota = \mathbf{R}$ , the set of real numbers, and  $I$  assigns  $+$ ,  $0$ ,  $-$ ,  $\cdot$ ,  $1$ ,  $^{-1}$ ,  $\text{pos}$ ,  $<$ ,  $\leq$ ,  $\text{ub}$ ,  $\text{lub}$  their usual meanings.

## 4 Partial Functions

In this section and the next section we will examine six examples of statements from Spivak’s *Calculus* involving undefinedness. The examples in this section

are definitions of partial functions, and the examples in the next section involve limits that can be undefined. For each example, we will display how Spivak expresses it in *Calculus* followed by how it can be formalized in STTWU. We will assume that in the STTWU formalizations the variables  $f, g$  are of type  $(\iota \rightarrow \iota)$  and the variables  $a, l, m, x, \delta, \epsilon$  are of type  $\iota$ . The listed page numbers refer to the first edition [24] of *Calculus*.

Spivak devotes two chapters in *Calculus* to functions. He emphasizes that functions are often partial and discusses several examples of partial functions. He handles partial functions according to the traditional approach. In fact, he says “the symbol  $f(x)$  makes sense only for  $x$  in the domain of  $f$ ; for other  $x$  the symbol  $f(x)$  is not defined” [p. 38].

**Example 1.** This is a definition of a partial function in which the function’s domain is given explicitly.

Spivak:  $k(x) = \frac{1}{x} + \frac{1}{x-1}$ ,  $x \neq 0, 1$  [p. 39].

STTWU:  $\forall x . \text{if}(x \neq 0 \wedge x \neq 1, k(x) = \frac{1}{x} + \frac{1}{x-1}, k(x) \uparrow)$ .

The formalization of the definition in STTWU is very explicit but certainly more verbose than Spivak’s definition. A second formalization in STTWU as an equation is

$k = \lambda x . \text{if}(x \neq 0 \wedge x \neq 1, \frac{1}{x} + \frac{1}{x-1}, \perp_\iota)$ .

Although the second formalization directly identifies what is being defined and is somewhat more succinct, we prefer the first formalization because it more faithfully captures the form and meaning of Spivak’s definition.

**Example 2.** This is a shortened version of the previous example in which the function’s domain is implicit and, as Spivak says, “is understood to consist of all [real] numbers for which the definition makes any sense at all” [p. 39].

Spivak:  $k(x) = \frac{1}{x} + \frac{1}{x-1}$  [p. 39].

STTWU:  $\forall x . k(x) \simeq \frac{1}{x} + \frac{1}{x-1}$ .

Notice that in the formalization quasi-equality  $\simeq$  must be used instead of ordinary equality  $=$ . Spivak does not distinguish in *Calculus* between these two kinds of equality, but he is certainly aware of the important distinction between them. See his comment after problem 28 of Chapter 5 on p. 92.

This example illustrates that a partial function can be precisely described in STTWU, as in informal mathematics, without mentioning the domain of the function. The definition is thus more succinct than it would be if the domain were made explicit. The implicit domain can always be determined later *if necessary*.

**Example 3.** This example defines the quotient of two functions.

$$\text{Spivak: } \left(\frac{f}{g}\right)(x) = \frac{f(x)}{g(x)} \text{ [p. 41].}$$

$$\text{STTWU: } \forall f, g, x : \text{fun\_div}(f)(g)(x) \simeq \frac{f(x)}{g(x)}.$$

`fun_div` is a constant of type  $((\iota \rightarrow \iota) \rightarrow ((\iota \rightarrow \iota) \rightarrow (\iota \rightarrow \iota)))$  that denotes the quotient of two functions. Notice that the domain of `fun_div(f)(g)` is not explicitly defined, but it can be computed to be  $D_f \cap D_g \cap \{x : \mathbf{R} \mid g(x) \neq 0\}$ .

## 5 Limits

The notion of a limit of a function at a point in its domain of application is perhaps the most important concept in calculus. Spivak devotes an entire chapter to it. Since a limit does not always exist, this concept is a rich source of undefinedness.

**Example 4.** This is Spivak’s definition of a limit—which is actually formed from two definitions: (1) what it means for a function to approach a limit near a point [p. 78] and (2) what the notation  $\lim_{x \rightarrow a} f(x)$  denotes [p. 81]<sup>3</sup>.

Spivak:  $\lim_{x \rightarrow a} f(x)$  denotes the real number  $l$  such that, for every  $\epsilon > 0$ , there is some  $\delta > 0$  such that, for all  $x$ , if  $0 < |x - a| < \delta$ , then  $|f(x) - l| < \epsilon$  [pp. 78, 81].

$$\begin{aligned} \text{STTWU: } & \forall f, a . \text{lim}(f)(a) \simeq \\ & (ll . \\ & (\forall \epsilon . 0 < \epsilon \Rightarrow \\ & (\exists \delta . 0 < \delta \wedge \\ & (\forall x . (0 < \text{abs}(x - a) \wedge \text{abs}(x - a) < \delta) \Rightarrow \\ & \text{abs}(f(x) - l) < \epsilon))) \end{aligned}$$

`abs` denotes the absolute value function. Both the informal and formal definitions of a limit at a point are defined as the value of a definite description provide the value exists. Other limit concepts, such as the limit of a sequence, are defined by similar definition descriptions.

**Example 5.** This is a theorem about the limit of the quotient of the identity function and another function  $g$ .

Spivak: If  $\lim_{x \rightarrow a} g(x) = m$  and  $m \neq 0$ , then  $\lim_{x \rightarrow a} \left(\frac{1}{g}\right)(x) = \frac{1}{m}$  [Theorem 2(3), p. 84].

$$\text{STTWU: } \forall g, a, m . (\text{lim}(g)(a) = m \wedge m \neq 0) \Rightarrow \text{lim}(\text{fun\_div}(\text{id})(g))(a) = \frac{1}{m}.$$

<sup>3</sup> Spivak remarks that a “more logical symbol” like  $\lim_a f$  is “so infuriatingly rigid that almost no one has seriously tried to use it” [p. 81].

id is  $\lambda x . x$ , the identity function. Notice that there is no explicit mention of the definedness of the limit of  $g$  at  $x$  in either Spivak’s theorem or in its formalization in STTWU. This is because the hypothesis that the limit  $\lim(g)(a)$  equals  $m$  automatically implies that  $\lim(g)(a)$  is defined by the third principle of the traditional approach.

**Example 6.** This is the definition of the notion of a function being continuous at a point.

Spivak: The function  $f$  is continuous at  $a$  if  $\lim_{x \rightarrow a} f(x) = f(a)$  [p. 93].

STTWU:  $\forall f, a . \text{cont}(f)(a) \Leftrightarrow \lim(f)(a) = f(a)$ .

It is crucial to use equality  $=$  instead of quasi-equality  $\simeq$  because  $f$  is continuous at  $a$  only if  $\lim(f)(a)$  and  $f(a)$  are both defined and equal to each other.

## 6 Conclusion

In this paper we have presented STTWU, a version of simple type theory that directly formalizes the traditional approach to undefinedness. We have also formalized in STTWU several examples from Spivak’s *Calculus* involving undefinedness. The formalizations are exceedingly faithful in form and meaning to how the examples are expressed by Spivak. Moreover, the conciseness that comes from Spivak’s use of the traditional approach is fully preserved in the formalizations in STTWU.

It is our recommendation that logics with undefinedness, such as STTWU, be considered as the logical basic for mechanized mathematics systems. They are much closer to mathematical practice with respect to undefinedness than standard logics, and as IMPS has demonstrated, they can be effectively implemented.

## 7 Acknowledgments

The author would like to thank Jacques Carette for many valuable discussions on formalizing undefinedness, partial functions, and countless other aspects of mathematics.

## 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. 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. T. Burge. *Truth and Some Referential Devices*. PhD thesis, Princeton University, 1971.
6. T. Burge. Truth and singular terms. In K. Lambert, editor, *Philosophical Applications of Free Logic*, pages 189–204. Oxford University Press, 1991.
7. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
8. W. M. Farmer. A partial functions version of Church’s simple theory of types. *Journal of Symbolic Logic*, 55:1269–91, 1990.
9. W. M. Farmer. A simple type theory with partial functions and subtypes. *Annals of Pure and Applied Logic*, 64:211–240, 1993.
10. 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.
11. W. M. Farmer. Reasoning about partial functions with the aid of a computer. *Erkenntnis*, 43:279–294, 1995.
12. W. M. Farmer. STMM: A set theory for mechanized mathematics. *Journal of Automated Reasoning*, 26:269–289, 2001.
13. W. M. Farmer. The seven virtues of simple type theory. SQRL Report No. 18, McMaster University, 2003.
14. W. M. Farmer. A sound and complete proof system for STTWU. Technical Report No. CAS-04-01-WF, McMaster University, 2004.
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. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11:213–248, 1993.
17. 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.
18. S. Feferman. Polymorphic typed lambda-calculi in a type-free axiomatic framework. *Contemporary Mathematics*, 106:101–136, 1990.
19. 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.
20. L. G. Monk. PDLM: A Proof Development Language for Mathematics. Technical Report M86-37, The MITRE Corporation, Bedford, Massachusetts, 1986.
21. O. Müller and K. Slind. Treating partiality in a logic of total functions. *The Computer Journal*, 40:640–652, 1997.
22. D. L. Parnas. Predicate logic for software engineering. *IEEE Transactions on Software Engineering*, 19:856–861, 1993.
23. R. Schock. *Logics without Existence Assumptions*. Almqvist & Wiksells, Stockholm, Sweden, 1968.
24. M. Spivak. *Calculus*. W. A. Benjamin, 1967.
25. A. Stump. Subset types and partial functions. In F. Baader, editor, *Automated Deduction—CADE-19*, volume 2741 of *Lecture Notes in Computer Science*, pages 151–165. Springer-Verlag, 2003.