CADE-22 Tutorial

Logics with Undefinedness

William M. Farmer

Department of Computing and Software
McMaster University
Hamilton, Ontario, Canada

McGill University
Montreal, Quebec, Canada
2 August 2009

Outline

- Part 1: The Nature of Undefinedness.
- Part 2: First-Order Logic with Undefinedness.
- Part 3: Other Logics with Undefinedness.
- Part 4: Implementing a Logic with Undefinedness
- Conclusion.
- References.

Part 1

The Nature of Undefinedness

What is Undefinedness?

- 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 and computer science.
- Sources of undefinedness:
  1. Improper function applications:

     $\frac{17}{0}$, $\sqrt{-4}$, $\tan(\frac{\pi}{2})$, $\lim_{x \to 0} \sin(\frac{1}{x})$, $\mathsf{top}(\mathsf{empty\_stack})$

  2. Improper definite descriptions:

     "the x such that $x^2 = 4$"

  3. Improper indefinite descriptions:

     "some x such $x^2 = -4$"

Partial Functions

- A partial function f has:
  1. A domain of definition D_f consisting of the values at which it is defined.
  2. A domain of application D_f^* consisting of the values to which it may be applied.
- An application $f(a)$ is undefined if $a \notin D_f$.
- f is total if $D_f = D_f^*$ and strictly partial if $D_f \subset D_f^*$.
- There are two views of partial functions:
  1. A total function is a true function, while a strictly partial function is a broken function.
  2. A partial function is a general kind of a function, while a total function is a special kind of function.
- The first view is dominate in computer science, but the second view is dominate in mathematics.

Five Examples

1. The definition of the function that returns the top of a stack.
2. The definition of the division function $/ : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$ defined by

$$x/y = x * y^{-1}.$$

3. The formula "$f(2,3)$ is defined" where $f : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$ defined by

$$f(x,y) = \sqrt{\sqrt{x+y} - \sqrt{x-y}}.$$

4. The trigonometric identity

$$\tan(x+y) = \frac{\tan(x) + \tan(y)}{1 + \tan(x) * \tan(y)}.$$

5. The following theorem about limits: For all $f : \mathbf{R} \to \mathbf{R}$, $a \in \mathbf{R}$,

$$\lim_{x->a-} f(x) = \lim_{x->a+} f(x) \text{ implies } \lim_{x->a} f(x) \text{ exists.}$$

Definite Description

- A definite description is a term of the form

  "the x such that A holds".

- In logic, a definite description is often written as $\iota\, x \,.\, A$.
  - Example: $\max(s) = \iota\, m \,.\, m \in s \wedge (\forall\, x \,.\, x \in s \Rightarrow x \leq m)$.
- The use of definite description is quite common in mathematics.
- Definite descriptions naturally lead to undefined terms such as:
  - $\iota\, x \,.\, x \neq x$.
  - $\iota\, x \,.\, x^2 = 2$.
- Definite descriptions can be eliminated as shown by B. Russell in his famous 1905 paper "On Denoting".
- Definite description is thus a source of practical expressivity, but not theoretical expressivity.

Indefinite Description

- An indefinite description is a term of the form

  "some x such that A holds".

- In logic, an indefinite description is often written as $\epsilon\, x\, .\, A$.
- The use of indefinite description is common in mathematics and computer science.
- Indefinite descriptions naturally lead to undefined terms such as:
  - $\epsilon\, x\, .\, x \neq x$.
  - $\epsilon\, x\, .\, x^2 = -2$.
- Indefinite descriptions cannot be eliminated in the same way as definite descriptions.
- Indefinite description entails the axiom of choice.
- Universal and existential quantification can be defined using indefinite description.

The Trouble with Traditional Logics

- In traditional logics terms are always defined.
- This is true in particular in first-order logic and simple type theory.
- As a result, the use of partial functions and definite and indefinite description is problematic in traditional logics.
- There are many approaches to handling partial functions and undefinedness in traditional logics (see [3, 15, 17]).

Approach 1: Partial Functions as Relations

- An n-ary partial function f is represented by the $(n+1)$-ary relation that denotes the graph of f.
- Advantages:
  - Easy to implement.
  - Logic does not have be changed.
- Disadvantages:
  - Statements become very verbose.
  - Partial functions are handled differently from total functions.
  - Does not handled definite and indefinite descriptions.

Approach 1: Examples

- Example 1:

  $\mathsf{top} : \mathsf{Stack} \times \mathsf{Elt} \to \mathsf{Bool}.$

  $\forall\, s : \mathsf{Stack}, e : \mathsf{Elt}\ .$
  $\quad \mathsf{top}(s, e) \Leftrightarrow \exists\, s' : \mathsf{Stack}\ .\ s = \mathsf{push}(s', e).$

- Example 2:

  $\mathsf{div} : \mathbf{R} \times \mathbf{R} \times \mathbf{R} \to \mathsf{Bool}.$

  $\forall\, x, y, z : \mathbf{R}\ .$
  $\quad \mathsf{div}(x, y, z) \Leftrightarrow y \neq 0 \wedge z = x * y^{-1}.$

Approach 2: Domain Predicates

- For each partial function f there is a predicate dom_f such that $\mathrm{dom}_f(x)$ iff $x \in D_f$.
- In a formula, each application of f is guarded by a corresponding application of dom_f:
  - $\mathrm{dom}_f(a) \Rightarrow A(f(a))$.
- Advantages:
  - Easy to implement.
  - Logic does not have be changed.
  - Partial and total functions are handled in the same way.
- Disadvantages:
  - Statements become very verbose.
  - Does not handled definite and indefinite descriptions.

Approach 2: Examples

- Example 1:

  $\mathsf{top} : \mathsf{Stack} \to \mathsf{Elt}.$
  $\mathsf{dom}_{\mathsf{top}} : \mathsf{Stack} \to \mathsf{Bool}.$

  $\forall\, s : \mathsf{Stack}, e : \mathsf{Elt} \,.\, \mathsf{top}(\mathsf{push}(s, e)) = e.$
  $\forall\, s : \mathsf{Stack} \,.$
  $\quad \mathsf{dom}_{\mathsf{top}}(s) \Leftrightarrow \exists\, s' : \mathsf{Stack}, e : \mathsf{Elt} \,.\, s = \mathsf{push}(s', e).$

- Example 2:

  $/ : \mathbf{R} \times \mathbf{R} \to \mathbf{R}.$
  $\mathsf{dom}_{/} : \mathbf{R} \times \mathbf{R} \to \mathsf{Bool}.$

  $\forall\, x, y : \mathbf{R} \,.\, \mathsf{dom}_{-1}(y) \Rightarrow x/y = x * y^{-1}.$
  $\forall\, x, y : \mathbf{R} \,.\, \mathsf{dom}_{/}(x, y) \Leftrightarrow \mathsf{dom}_{-1}(y).$

Approach 3: Type Enforcement

- An application $f(a)$ of a function f to an argument a is well-formed only if $a \in D_f$.
- This is enforced with a type system: $f(a)$ is well-formed only if $f(a)$ is type correct.
- Partial functions are thus treated as total functions on a restricted type.
  - For example, the type of / would be $\mathbf{R} \times (\mathbf{R} \setminus \{0\})$.
- Advantages:
  - Partial functions are effectively total functions.
  - Can be used with some traditional logics.
- Disadvantages:
  - Requires a sophisticated type system.
  - Type checking is undecidable.
  - Does not handled definite and indefinite descriptions.

Approach 3: Examples

- Example 1:

  $\mathsf{push} : \mathsf{Stack} \times \mathsf{Elt} \to \mathsf{PushStack}$
  $\mathsf{top} : \mathsf{PushStack} \to \mathsf{Elt}.$

  $\forall\, s : \mathsf{Stack}, e : \mathsf{Elt} \;.\; \mathsf{top}(\mathsf{push}(s, e)) = e.$

- Example 2:

  $/ : \mathbf{R} \times \mathbf{R}^{-0} \to \mathbf{R}.$

  $\forall\, x : \mathbf{R}, y : \mathbf{R}^{-0} \;.\; x/y = x * y^{-1}.$

Approach 4: Unspecified Values

- A term is considered undefined if its value is completely unspecified.
  - Thus an undefined term has some value, but nothing can be said about it.
- Advantages:
  - Easy to implement.
  - Logic does not have to be changed.
  - Handles both partial functions and definite and indefinite descriptions.
- Disadvantages:
  - Undefinedness can only be expressed implicitly.
  - Statements often require the use of definedness conditions.

Approach 4: Examples

- Example 1:

  $\mathsf{top} : \mathsf{Stack} \to \mathsf{Elt}.$

  $\forall\, s : \mathsf{Stack}, e : \mathsf{Elt} \,.\, \mathsf{top}(\mathsf{push}(s, e)) = e.$

- Example 2:

  $/ : \mathbf{R} \times \mathbf{R} \to \mathbf{R}.$

  $\forall\, x, y : \mathbf{R} \,.\, x/y = x * y^{-1}.$

Approach 5: Internal Error Values

- The value of an undefined term of type α is some error value $e_\alpha \in D_\alpha$.
- Advantages:
  - Logic does not have be changed.
  - Handles both partial functions and definite and indefinite descriptions.
- Disadvantages:
  - Often definedness conditions are needed to distinguish between when e_α means undefined and when e_α means the value of D_α.
  - Need a different error element for each type.

Approach 5: Examples

- Example 1:

  $\mathsf{top} : \mathsf{Stack} \to \mathsf{Elt}.$

  $\forall\, s : \mathsf{Stack}, e : \mathsf{Elt} \,.\, \mathsf{top}(\mathsf{push}(s, e)) = e.$
  $\mathsf{top}(\mathsf{empty\_stack}) = -1.$

- Example 2:

  $/ : \mathbf{R} \times \mathbf{R} \to \mathbf{R}.$

  $\forall\, x, y : \mathbf{R} \,.\, y \neq 0 \Rightarrow x/y = x * y^{-1}.$
  $\forall\, x : \mathbf{R} \,.\, x/0 = 0.$

Approach 6: External Error Values

- For each type α, an error value $\bot_\alpha$ is added to α to produce a new type $\alpha_\bot$.
  - $D_{\alpha_\bot} = D_\alpha \cup \{\bot_\alpha\}$, and α is thus a subtype of $\alpha_\bot$.
- A term of type $\alpha_\bot$ is undefined if its value is $\bot_\alpha$.
- Advantages:
  - Error values are separate from other values.
  - Handles both partial functions and definite and indefinite descriptions.
- Disadvantages:
  - Logic must be modified to include error values.
  - Quantification very often needs to be restricted to the nonerror values of a type.

Approach 6: Examples

- Example 1:

  $\mathsf{top} : \mathsf{Stack}_\bot \to \mathsf{Elt}_\bot.$

  $\forall\, s : \mathsf{Stack}, e : \mathsf{Elt}\ .\ \mathsf{top}(\mathsf{push}(s, e)) = e.$
  $\mathsf{top}(\mathsf{empty\_stack}) = \bot_{\mathsf{Elt}}.$
  $\mathsf{top}(\bot_{\mathsf{Stack}}) = \bot_{\mathsf{Elt}}.$

- Example 2:

  $/ : \mathbf{R}_\bot \times \mathbf{R}_\bot \to \mathbf{R}_\bot.$

  $\forall\, x, y : \mathbf{R}\ .\ y \neq 0 \Rightarrow x/y = x * y^{-1}.$
  $\forall\, x : \mathbf{R}\ .\ x/0 = \bot_{\mathbf{R}}.$
  $\forall\, x, y : \mathbf{R}_\bot\ .\ (x = \bot_{\mathbf{R}} \vee y = \bot_{\mathbf{R}}) \Rightarrow x/y = \bot_{\mathbf{R}}.$

Traditional Approach to Undefinedness

There is a traditional approach to undefinedness [6, 8] employed in mathematical practice that is based on three principles:

1. Atomic terms (i.e., variables and constants) are always defined.
2. Compound terms may be undefined.
   - A function application $f(a)$ is undefined if f is undefined, a is undefined, or $a \notin \mathrm{dom}(f)$.
   - 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.
   - An indefinite description "some x that has property P" is undefined if there is no x that has property P.
3. Formulas are always true or false and hence are always defined.
   - A predicate application $p(a)$ is false if p is undefined, a is undefined, or $a \notin \mathrm{dom}(p)$.

Benefits of the Traditional Approach

- Meaningful statements can include undefined terms.

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

- Function domains can be implicit.

  $k(x) \simeq \frac{1}{x} + \frac{1}{x-1}.$
  $\left(\frac{f}{g}\right)(x) \simeq \frac{f(x)}{g(x)}.$

- Definedness assumptions can be implicit, and as a result, expressions involving undefinedness can be very concise.

  $\forall\, x, y, z : \mathbf{R} \,.\, \frac{x}{y} = z \Rightarrow x = y * z.$

- Improper function applications and definite and indefinite descriptions are handled in the same way.

Formalizations of the Traditional Approach

- Several logicians, computer scientists, and software engineers have independently proposed logics that are essentially formalizations of the traditional approach to undefinedness [1, 2, 16, 18, 19].
- These logics have been obtained by slightly modifying traditional logics such as first-order logic and simple type theory.

Part 2

First-Order Logic with Undefinedness

Two Versions of First-Order Logic

- FOL is a version of traditional first-order logic.
  - All terms are defined.
  - All functions are total.
- FOL with Undefinedness (FOLwU) is a version of first-order logic that is obtained by slightly modifying FOL.
  - Formalizes the traditional approach to undefinedness.
  - New machinery is convenient, but not essential.
  - See [1, 13] for details.

Syntax of FOL: Languages

- Let $\mathcal{V}$ be a fixed infinite set of symbols called variables.
- A language of FOL is a triple $L = (\mathcal{C}, \mathcal{F}, \mathcal{P})$ where:
  - $\mathcal{C}$ is a set of symbols called individual constants.
  - $\mathcal{F}$ is a set of symbols called function symbols, each with an assigned arity ≥ 1.
  - $\mathcal{P}$ is a set of symbols called predicate symbols, each with an assigned arity ≥ 1. $\mathcal{P}$ contains the binary predicate symbol $=$.
  - $\mathcal{V}$, $\mathcal{C}$, $\mathcal{F}$, and $\mathcal{P}$ are pairwise disjoint.

Syntax of FOL: Terms and Formulas

- Let $L = (\mathcal{C}, \mathcal{F}, \mathcal{P})$ be a language of FOL.
- A term of L is a string of symbols inductively defined by the following formation rules:
  - Each $x \in \mathcal{V}$ and $a \in \mathcal{C}$ is a term of L.
  - If $f \in \mathcal{F}$ is n-ary and $t_1, \ldots, t_n$ are terms of L, then $f(t_1, \ldots, t_n)$ is a term of L.
- A formula of L is a string of symbols inductively defined by the following formation rules:
  - If $p \in \mathcal{P}$ is n-ary and $t_1, \ldots, t_n$ are terms of L, then $p(t_1, \ldots, t_n)$ is a formula of L.
  - If A and B are formulas of L and $x \in \mathcal{V}$, then $(\neg A)$ and $(A \Rightarrow B)$, and $(\forall x \,.\, A)$ are formulas of L.
- $=$, $\neg$, $\Rightarrow$, and $\forall$ are the logical constants of FOL.

Syntax of FOL: Notational Definitions

$(s = t)$	denotes	$= (s, t)$.
$(s \neq t)$	denotes	$(\neg(s = t))$.
T	denotes	$(\forall\, x \,.\, (x = x))$.
F	denotes	$(\neg(\mathsf{T}))$.
$(A \vee B)$	denotes	$((\neg A) \Rightarrow B)$.
$(A \wedge B)$	denotes	$(\neg((\neg A) \vee (\neg B)))$.
$(A \Leftrightarrow B)$	denotes	$((A \Rightarrow B) \wedge (B \Rightarrow A))$.
$(\exists\, x \,.\, A)$	denotes	$(\neg(\forall\, x \,.\, (\neg A)))$.

Semantics of FOL: Models

- A model for a language $L = (\mathcal{C}, \mathcal{F}, \mathcal{P})$ of FOL is a pair $M = (D, I)$ where D is a nonempty domain (set) and I is a total function on $\mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$ such that:
  1. If $a \in \mathcal{C}$, $I(a) \in D$.
  2. If $f \in \mathcal{F}$ is n-ary, $I(f) : D^n \to D$ and $I(f)$ is total.
  3. If $p \in \mathcal{P}$ is n-ary, $I(p) : D^n \to \{\mathrm{T}, \mathrm{F}\}$ and $I(p)$ is total.
  4. $I(=)$ is id_D, the identity predicate on D.
- A variable assignment into M is a function that maps each $x \in \mathcal{V}$ to an element of D.
- Given a variable assignment φ into M, $x \in \mathcal{V}$, and $d \in D$, let $\varphi[x \mapsto d]$ be the variable assignment φ' into M such $\varphi'(x) = d$ and $\varphi'(y) = \varphi(y)$ for all $y \neq x$.

Semantics of FOL: Valuation Function (1/2)

The valuation function for a model M for a language $L = (\mathcal{C}, \mathcal{F}, \mathcal{P})$ of FOL is the total binary function V^M that satisfies the following conditions for all variable assignments φ into M and all terms t and formulas A of L:

1. Let $t \in \mathcal{V}$. Then $V^M_\varphi(t) = \varphi(t)$.
2. Let $t \in \mathcal{C}$. Then $V^M_\varphi(t) = I(t)$.
3. Let $t = f(t_1, \ldots, t_n)$. Then

$$V^M_\varphi(t) = I(f)(V^M_\varphi(t_1), \ldots, V^M_\varphi(t_n)).$$

Semantics of FOL: Valuation Function (2/2)

4. Let $A = p(t_1, \ldots, t_n)$. Then

$$V_\varphi^M(A) = I(p)(V_\varphi^M(t_1), \ldots, V_\varphi^M(t_n)).$$

5. Let $A = (\neg A')$. If $V_\varphi^M(A') = \mathrm{F}$, then $V_\varphi^M(A) = \mathrm{T}$; otherwise $V_\varphi^M(A) = \mathrm{F}$.
6. Let $A = (A_1 \Rightarrow A_2)$. If $V_\varphi^M(A_1) = \mathrm{T}$ and $V_\varphi^M(A_2) = \mathrm{F}$, then $V_\varphi^M(A) = \mathrm{F}$; otherwise $V_\varphi^M(A) = \mathrm{T}$.
7. Let $A = (\forall\, x \,.\, A')$. If $V_{\varphi[x \mapsto d]}^M(A') = \mathrm{T}$ for all $d \in D$, then $V_\varphi^M(A) = \mathrm{T}$; otherwise $V_\varphi^M(A) = \mathrm{F}$.

Proof System of FOL: Axiom Schemas

1. $A \Rightarrow (B \Rightarrow A)$.
2. $(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$.
3. $(\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A)$.
4. $(\forall\, x \,.\, A \Rightarrow B) \Rightarrow (A \Rightarrow \forall\, x \,.\, B)$ where x is not free in A.
5. $(\forall\, x \,.\, A) \Rightarrow A[x \mapsto t]$ where t is free for x in A.
6. $\forall\, x \,.\, x = x$.
7. $s = t \Rightarrow (A \Rightarrow A^*)$ where A^* is the result of replacing one occurrence of s in A with t, provided that the occurrence of s is not a variable immediately after $\forall$.

Proof System of FOL: Rules of Inference

1. From A and $A \Rightarrow B$ infer B.
2. From A infer $\forall\, x \,.\, A$.

Syntax of FOLwU: Languages

- Let $\mathcal{V}$ be a fixed infinite set of symbols called variables.
- A language of FOLwU is a triple $L = (\mathcal{C}, \mathcal{F}, \mathcal{P})$ where:
  - $\mathcal{C}$ is a set of symbols called individual constants.
  - $\mathcal{F}$ is a set of symbols called function symbols, each with an assigned arity ≥ 1.
  - $\mathcal{P}$ is a set of symbols called predicate symbols, each with an assigned arity ≥ 1. $\mathcal{P}$ contains the binary predicate symbol $=$.
  - $\mathcal{V}$, $\mathcal{C}$, $\mathcal{F}$, and $\mathcal{P}$ are pairwise disjoint.

Syntax of FOLwU: Terms and Formulas

- Let $L = (\mathcal{C}, \mathcal{F}, \mathcal{P})$ be a language of FOLwU.
- A term of L is a string of symbols inductively defined by the following formation rules:
  - Each $x \in \mathcal{V}$ and $a \in \mathcal{C}$ is a term of L.
  - If $f \in \mathcal{F}$ is n-ary and $t_1, \ldots, t_n$ are terms of L, then $f(t_1, \ldots, t_n)$ is a term of L.
  - If $x \in \mathcal{V}$ and A is a formula of L, then $(\mathrm{I}\, x \,.\, A)$ is a term of L.
- A formula of L is a string of symbols inductively defined by the following formation rules:
  - If $p \in \mathcal{P}$ is n-ary and $t_1, \ldots, t_n$ are terms of L, then $p(t_1, \ldots, t_n)$ is a formula of L.
  - If A and B are formulas of L and $x \in \mathcal{V}$, then $(\neg A)$ and $(A \Rightarrow B)$, and $(\forall\, x \,.\, A)$ are formulas of L.
- $=$, $\neg$, $\Rightarrow$, $\forall$, I are the logical constants of FOLwU.

Syntax of FOLwU: Notational Definitions

$(s = t)$	denotes	$= (s, t)$.
$(s \neq t)$	denotes	$(\neg(s = t))$.
T	denotes	$(\forall\, x \,.\, (x = x))$.
F	denotes	$(\neg(\mathsf{T}))$.
$(A \vee B)$	denotes	$((\neg A) \Rightarrow B)$.
$(A \wedge B)$	denotes	$(\neg((\neg A) \vee (\neg B)))$.
$(A \Leftrightarrow B)$	denotes	$((A \Rightarrow B) \wedge (B \Rightarrow A))$.
$(\exists\, x \,.\, A)$	denotes	$(\neg(\forall\, x \,.\, (\neg A)))$.
$(t\downarrow)$	denotes	$\exists\, x \,.\, x = t$
		where x does not occur in t
$(t\uparrow)$	denotes	$\neg(t\downarrow)$.
$(s \simeq t)$	denotes	$(s\downarrow \vee\, t\downarrow) \Rightarrow s = t$
$\bot$	denotes	$\mathrm{I}\, x \,.\, x \neq x$
$\mathsf{if}(A, s, t)$	denotes	$\mathrm{I}\, x \,.\, (A \Rightarrow x = s) \vee (\neg A \Rightarrow x = t)$
		where x does not occur in A, s, or t

Semantics of FOLwU: Models

- A model for a language $L = (\mathcal{C}, \mathcal{F}, \mathcal{P})$ of FOLwU is a pair $M = (D, I)$ where D is a nonempty domain (set) and I is a total function on $\mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$ such that:
  1. If $a \in \mathcal{C}$, $I(a) \in D$.
  2. If $f \in \mathcal{F}$ is n-ary, $I(f) : D^n \to D$ and $I(f)$ is partial.
  3. If $p \in \mathcal{P}$ is n-ary, $I(p) : D^n \to \{\mathrm{T}, \mathrm{F}\}$ and $I(p)$ is total.
  4. $I(=)$ is id_D, the identity predicate on D.
- A variable assignment into M is a function that maps each $x \in \mathcal{V}$ to an element of D.
- Given a variable assignment φ into M, $x \in \mathcal{V}$, and $d \in D$, let $\varphi[x \mapsto d]$ be the variable assignment φ' into M such $\varphi'(x) = d$ and $\varphi'(y) = \varphi(y)$ for all $y \neq x$.

Semantics of FOLwU: Valuation Function (1/2)

The valuation function for a model M for a language $L = (\mathcal{C}, \mathcal{F}, \mathcal{P})$ of FOLwU is the partial binary function V^M that satisfies the following conditions for all variable assignments φ into M and all terms t and formulas A of L:

1. Let $t \in \mathcal{V}$. Then $V^M_\varphi(t) = \varphi(t)$.
2. Let $t \in \mathcal{C}$. Then $V^M_\varphi(t) = I(t)$.
3. Let $t = f(t_1, \ldots, t_n)$. If $V^M_\varphi(t_1), \ldots, V^M_\varphi(t_n)$ are defined and $I(f)$ is defined at $(V^M_\varphi(t_1), \ldots, V^M_\varphi(t_n))$, then

$$V^M_\varphi(t) = I(f)(V^M_\varphi(t_1), \ldots, V^M_\varphi(t_n)).$$

   Otherwise $V^M_\varphi(t)$ is undefined.
4. Let $t = (\mathrm{I}\, x \,.\, A)$. If $V^M_{\varphi[x \mapsto d]}(A) = \mathrm{T}$ for a unique $d \in D$, then $V^M_\varphi(t) = d$. Otherwise $V^M_\varphi(t)$ is undefined.

Semantics of FOLwU: Valuation Function (2/2)

5. Let $A = p(t_1, \ldots, t_n)$. If $V_\varphi^M(t_1), \ldots, V_\varphi^M(t_n)$ are defined, then

$$V_\varphi^M(A) = I(p)(V_\varphi^M(t_1), \ldots, V_\varphi^M(t_n)).$$

   Otherwise, $V_\varphi^M(A) = \mathrm{F}$.

6. Let $A = (\neg A')$. If $V_\varphi^M(A') = \mathrm{F}$, then $V_\varphi^M(A) = \mathrm{T}$; otherwise $V_\varphi^M(A) = \mathrm{F}$.
7. Let $A = (A_1 \Rightarrow A_2)$. If $V_\varphi^M(A_1) = \mathrm{T}$ and $V_\varphi^M(A_2) = \mathrm{F}$, then $V_\varphi^M(A) = \mathrm{F}$; otherwise $V_\varphi^M(A) = \mathrm{T}$.
8. Let $A = (\forall\, x \,.\, A')$. If $V_{\varphi[x \mapsto d]}^M(A') = \mathrm{T}$ for all $d \in D$, then $V_\varphi^M(A) = \mathrm{T}$; otherwise $V_\varphi^M(A) = \mathrm{F}$.

Proof System of FOLwU: Axiom Schemas

1. $A \Rightarrow (B \Rightarrow A)$.
2. $(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$.
3. $(\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A)$.
4. $(\forall\, x \,.\, A \Rightarrow B) \Rightarrow (A \Rightarrow \forall\, x \,.\, B)$ where x is not free in A.
5. $((\forall\, x \,.\, A) \wedge t\!\downarrow) \Rightarrow A[x \mapsto t]$ where t is free for x in A.
6. $\forall\, x \,.\, x = x$.
7. $s \simeq t \Rightarrow (A \Rightarrow A^*)$ where A^* is the result of replacing one occurrence of s in A with t, provided that the occurrence of s is not a variable immediately after $\forall$ or I.

Proof System of FOLwU: Axiom Schemas

8. $x\downarrow$ where $x \in \mathcal{V}$.
9. $c\downarrow$ where $c \in \mathcal{C}$.
10. $(t_1\uparrow \vee \cdots \vee t_n\uparrow) \Rightarrow f(t_1, \ldots, t_n)\uparrow$ where $f \in \mathcal{F}$ is n-ary.
11. $(t_1\uparrow \vee \cdots \vee t_n\uparrow) \Rightarrow \neg p(t_1, \ldots, t_n)$ where $p \in \mathcal{P}$ is n-ary.
12. $(\exists\, !\, x \,.\, A) \Rightarrow ((\mathrm{I}\, x \,.\, A)\downarrow \wedge A[x \mapsto (\mathrm{I}\, x \,.\, A)])$ where $(\mathrm{I}\, x \,.\, A)$ is free for x in A.
13. $\neg(\exists\, !\, x \,.\, A) \Rightarrow (\mathrm{I}\, x \,.\, A)\uparrow$.

Proof System of FOLwU: Rules of Inference

1. From A and $A \Rightarrow B$ infer B.
2. From A infer $\forall\, x \,.\, A$.

FOLwU: Five Examples

1. $\forall\, s \,.\, \mathsf{top}(s) \simeq \mathrm{I}\, e \,.\, \exists\, s' \,.\, s = \mathsf{push}(s', e)$.
2. $\forall\, x, y \,.\, x/y \simeq x * y^{-1}$.

   $\forall\, x, y \,.\, x/y \simeq \mathrm{I}\, z \,.\, x = y * z$.
3. $f(2,3)\!\downarrow$ where

   $$\forall\, x, y \,.\, f(x,y) \simeq \sqrt{\sqrt{x+y} - \sqrt{x-y}}.$$
4. $\forall\, x, y \,.\, \mathsf{tan}(x+y)\!\downarrow \;\wedge\; \frac{\mathsf{tan}(x)+\mathsf{tan}(y)}{1+\mathsf{tan}(x)*\mathsf{tan}(y)}\!\downarrow \;\Rightarrow$

   $$\mathsf{tan}(x+y) = \frac{\mathsf{tan}(x)+\mathsf{tan}(y)}{1+\mathsf{tan}(x)*\mathsf{tan}(y)}.$$
5. $\forall\, x \,.\, \lim_{x->a-} f(x) = \lim_{x->a+} f(x) \Rightarrow (\lim_{x->a} f(x))\!\downarrow$.

The Elimination Theorem

The machinery in FOLwU for partial functions and undefined terms is purely a convenience — it can freely eliminated.

Elimination Theorem. For every theory T of FOLwU, there is a theory T^* of FOL and a translation from each formula A of T to a formula A^* of T^* such that:

$$T \models A \text{ iff } T^* \models A^*.$$

Proof. First, eliminate each n-ary function symbol f by replacing it with a $(n+1)$-ary predicate symbol p_f that denotes its graph. For example, $p(s, f(t))$ is replaced with $\exists y \,.\, p_f(t, y) \wedge p(s, y)$. Second, eliminate each occurrence of $\mathrm{I}\, x \,.\, A$ by replacing it with a unique existentially quantified variable. For example, $p(s, (\mathrm{I}\, x \,.\, A))$ is replaced with $\exists !\, x \,.\, A \wedge p(s, x)$.

Notes about FOLwU

- Formalizes the traditional approach to undefinedness.
  - Functions are strict with respect to undefinedness.
  - Predicates are strict with respect to undefinedness.
- The definedness and undefinedness of a term can be directly stated: $t\downarrow$, $t\uparrow$.
- Undefined is like a universal external error value.
  - All undefined terms have the same "value".
- Undefined is not a genuine value.
  - Undefined cannot be an input to a function or a predicate.
  - The range of a variable does not included undefined.
- Undefined terms are indiscernible.

Disadvantages of FOLwU

- FOLwU is not a traditional logic.
- FOLwU is moderately harder to implement than FOL.

Advantages of FOLwU

- Formalizes the traditional approach to undefinedness:
  - Expressions that include undefined terms can be meaningful.
  - Function domains can be implicit.
  - Definedness assumptions can be implicit.
  - Improper function applications and definite and indefinite descriptions are handled in the same way.
- Definedness checking may be performed only as needed.
- Closer to standard mathematical practice than FOL.
- Has same theoretical expressivity as FOL, but much greater practical expressivity.

Part 3

Other Logics with Undefinedness

Simple Type Theory

- The traditional approach to undefinedness can be formalized in simple type theory in much the same way as in first-order logic [3].
- In [11] we present a modification of Peter Andrews' formulation of Church's type theory that directly formalizes the traditional approach of undefinedness.
- A version of Church's type theory with undefinedness named LUTINS [4, 5] is the logic of the IMPS theorem proving system [14].
- STTwU [8] is a version of STT [12],a simple version of simple type theory, with undefinedness.

IMPS

- IMPS is a higher-order theorem proving system developed at The MITRE Corporation by W. Farmer, J. Guttman, and J. Thayer.
- Distinguishing characteristics:
  1. Logic is simple type theory with undefinedness and subtypes.
  2. Little theories method for organizing mathematics.
  3. Proofs that combine deduction and computation.
- The IMPS theory library contains significant portions of analysis as well as algebra and logic.
- Available without fee by public license (first released in 1993).
  - Written in T and Common Lisp.
  - User interface based on XEmacs.
  - Minimally maintained.
- IMPS web page is at

  `http:imps.mcmaster.ca/`.

Syntax of STTwU: Types

- A type of STTwU is a string of symbols defined by the following formation rules:

$$\textbf{T1}\ \frac{}{\mathbf{type}[\iota]}\ \text{(Type of individuals)}$$

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

$$\textbf{T3}\ \frac{\mathbf{type}[\alpha],\ \mathbf{type}[\beta]}{\mathbf{type}[(\alpha \to \beta)]}\ \text{(Function type)}$$

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

Syntax of STTwU: Symbols

- The logical symbols of STTwU are:
  - Function application: @ (hidden).
  - Function abstraction: λ.
  - Equality: $=$.
  - Definite description: I (capital iota).
  - An infinite set $\mathcal{V}$ of symbols called variables.
- A language of STTwU is a pair $L = (\mathcal{C}, \tau)$ where:
  - $\mathcal{C}$ is a set of symbols called constants.
  - $\mathcal{V}$ and $\mathcal{C}$ are disjoint.
  - $\tau : \mathcal{C} \to \mathcal{T}$ is a total function.

Syntax of STTwU: Expressions

An expression E of type α of a STTwU language $L = (\mathcal{C}, \tau)$ is defined by the following rules:

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

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

E3 $$\frac{\mathbf{expr}_L[A, \alpha],\ \mathbf{expr}_L[F, (\alpha \to \beta)]}{\mathbf{expr}_L[F(A), \beta]}$$ (Application)

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

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

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

Notational Definitions

T	denotes	$(\lambda\, x : * \,.\, x) = (\lambda\, x : * \,.\, x)$
F	denotes	$(\lambda\, x : * \,.\, \mathsf{T}) = (\lambda\, x : * \,.\, x)$
$(\neg A_*)$	denotes	$A_* = \mathsf{F}$
$(A_* \wedge B_*)$	denotes	$(\lambda\, f : * \to (* \to *) \,.\, f(\mathsf{T})(\mathsf{T})) =$ $(\lambda\, f : * \to (* \to *) \,.\, f(A_*)(B_*))$
$(A_* \vee B_*)$	denotes	$\neg(\neg A_* \wedge \neg B_*)$
$(A_* \Rightarrow B_*)$	denotes	$\neg A_* \vee B_*$
$(\forall\, x : \alpha \,.\, A_*)$	denotes	$(\lambda\, x : \alpha \,.\, A_*) = (\lambda\, x : \alpha \,.\, \mathsf{T})$
$(\exists\, x : \alpha \,.\, A_*)$	denotes	$\neg(\forall\, x : \alpha \,.\, \neg A_*)$
$(A_\alpha\downarrow)$	denotes	$\exists\, x : \alpha \,.\, x = A_\alpha$
$(A_\alpha\uparrow)$	denotes	$\neg(A_\alpha\downarrow)$
$(A_\alpha \simeq B_\alpha)$	denotes	$(A_\alpha\downarrow \vee B_\alpha\downarrow) \Rightarrow A_\alpha = B_\alpha$
$\bot_\alpha$	denotes	$\mathrm{I}\, x : \alpha \,.\, x \neq x$
$\mathsf{if}(A_*, B_\alpha, C_\alpha)$	denotes	$\mathrm{I}\, x : \alpha \,.\, (A_* \Rightarrow x = B_\alpha) \wedge$ $(\neg A_* \Rightarrow x = C_\alpha)$

Semantics of STTwU: Standard Models

- A standard model for a language $L = (\mathcal{C}, \tau)$ of STT is a triple $M = (\mathcal{D}, I)$ where:
  - $\mathcal{D} = \{D_\alpha : \alpha \in \mathcal{T}\}$ is a set of nonempty domains (sets).
  - $D_* = \{\mathrm{T}, \mathrm{F}\}$, the domain of truth values.
  - $D_{\alpha\to\beta}$ is the set of all partial functions from D_α to D_β.
  - I maps each $c \in \mathcal{C}$ to an element of $D_{\tau(c)}$.
- A variable assignment into M is a function that maps each expression $(x : \alpha)$ to an element of D_α.
- Given a variable assignment φ into M, an expression $(x : \alpha)$, and $d \in D_\alpha$, let $\varphi[(x : \alpha) \mapsto d]$ be the variable assignment φ' into M such that $\varphi'((x : \alpha)) = d$ and $\varphi'(v) = \varphi(v)$ for all $v \neq (x : \alpha)$.

Semantics of STTwU: Valuation Function (1/2)

The valuation function for a standard model $M = (\mathcal{D}, I)$ for a language $L = (\mathcal{C}, \tau)$ of STTwU is the partial binary function V^M that satisfies the following conditions for all variable assignments φ into M and all expressions E of L:

1. Let E is $(x : \alpha)$. Then $V^M_\varphi(E) = \varphi(E)$.
2. Let $E \in \mathcal{C}$. Then $V^M_\varphi(E) = I(E)$.
3. Let E_β be $F(A)$. If $V^M_\varphi(F)$ is defined, $V^M_\varphi(A)$ is defined, and $V^M_\varphi(A)$ is in the domain of $V^M_\varphi(F)$, then

$$V^M_\varphi(E_\alpha) = V^M_\varphi(F)(V^M_\varphi(A)).$$

Otherwise, $V^M_\varphi(E_\beta) = \mathrm{F}$ if $\beta = *$ and $V^M_\varphi(E_\beta)$ is undefined if $\beta \neq *$.

Semantics of STTwU: Valuation Function (2/2)

4. Let $E_{\alpha\to\beta}$ be $(\lambda\, x : \alpha \;.\; B)$. Then $V^M_\varphi(E_{\alpha\to\beta})$ is the partial function $f : D_\alpha \to D_\beta$ such that, for each $d \in D_\alpha$, $f(d) = V^M_{\varphi[(x:\alpha)\mapsto d]}(B)$ if $V^M_{\varphi[(x:\alpha)\mapsto d]}(B)$ is defined and $f(d)$ is undefined if $V^M_{\varphi[(x:\alpha)\mapsto d]}(B)$ is undefined.
5. Let E_* be $(E_1 = E_2)$. If $V^M_\varphi(E_1)$ is defined, $V^M_\varphi(E_2)$ is defined, and $V^M_\varphi(E_1) = V^M_\varphi(E_2)$, then $V^M_\varphi(E_*) = \mathrm{T}$. Otherwise $V^M_\varphi(E_*) = \mathrm{F}$.
6. Let E_α be $(\mathrm{I}\, x : \alpha \;.\; A)$. If there is a unique $d \in D_\alpha$ such that $V^M_{\varphi[(x:\alpha)\mapsto d]}(A) = \mathrm{T}$, then $V^M_\varphi(E_\alpha) = d$. Otherwise, $V^M_\varphi(E_\alpha) = \mathrm{F}$.

STTwU Proof System: Axiom Schemas (1/4)

A1 (Truth Values)

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

A2 (Leibniz' Law)

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

A3 (Extensionality)

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

A4 (Beta-Reduction)

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

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

STTwU Proof System: Axiom Schemas (2/4)

A5 (Variables are Defined)

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

A6 (Constants are Defined)

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

A7 (Function Abstractions are Defined)

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

A8 (Predicate Applications are Defined)

$F_{\alpha\to *}(A_\alpha)\downarrow$.

STTwU Proof System: Axiom Schemas (3/4)

A9 (Improper Predicate Application)

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

A10 (Improper Function Application)

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

A11 (Improper Equality)

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

A12 (Equality and Quasi-Quality)

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

STTwU Proof System: Axiom Schemas (4/4)

A13 (Proper Definite Description)

$$(\exists\,!\,x : \alpha \;.\; A_*) \Rightarrow$$

$$((\mathrm{I}\,x : \alpha \;.\; A_*)\downarrow \wedge A_*[(x : \alpha) \mapsto (\mathrm{I}\,x : \alpha \;.\; A_*)])$$

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

A14 (Improper Definite Description)

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

STTwU Proof System: Rules of Inference

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_α in C_* by an occurrence of B_α, provided that the occurrence of A_α in C_* is not immediately preceded by λ.

Simple Type Theory with Subtypes

- The type system of a logic like STTwU can be extended to admit subtypes.
  - $\alpha \ll \beta$ means $D_\alpha \subseteq D_\beta$.
  - Types are covariant with respect $\rightarrow$:
    If $\alpha \ll \alpha'$ and $\beta \ll \beta'$, then $(\alpha \rightarrow \beta) \ll (\alpha' \rightarrow \beta')$.
  - Examples: $\mathbf{N} \ll \mathbf{Z} \ll \mathbf{Q} \ll \mathbf{R}$ and $(\mathbf{N} \rightarrow \mathbf{Z}) \ll (\mathbf{R} \rightarrow \mathbf{Q})$.
- The IMPS logic LUTINS is an example of a simple type theory with undefinedness and subtypes.
  - Subtypes are called sorts in IMPS.
- Every expression is assigned one nominal type but can "reside" in many types.
  - $(A_\alpha \downarrow \beta)$ denotes $\exists\, x : \beta \,.\, x = A_\alpha$.
  - $(A_\alpha \downarrow)$ denotes $(A_\alpha \downarrow \alpha)$.
- Subtypes are quite convenient for expressing mathematics.

NBG Set Theory

- STMM [7, 13] is a version of NBG set theory intended to be a Set Theory for Mechanized Mathematics.
- NBG set theory admits proper classes as well as sets.
  - The universe of sets is a proper class.
  - Total functions like the cardinality function is a proper class.
- STMM is a logic with undefinedness.
- STMM has a type system with a universal type, function types, and subtypes.
- Definedness checking includes checking for whether a term denotes a set (as opposed to a proper class).

Chiron

- Chiron [9, 10] is a logic based on NBG set theory that is intended to be a practical, general-purpose logic for mechanizing mathematics.
- It is a logic with undefinedness.
- It has a type system with a universal type, dependent types, dependent function types, subtypes, and possibly empty types.
- It has a facility for reasoning about the syntax of expressions that employs quotation and evaluation.
- Ungrounded terms are treated as being undefined, and ungrounded formulas (like the liar paradox) are treated as being false.

Part 4

Implementing a Logic with Undefinedness

Overview

- Implementing a logic with undefinedness is very similar to implementing the corresponding traditional logic.
- There are three key laws that must be implemented differently:
  1. Substitution for a variable.
  2. Equality substitution.
  3. Definition of a constant.
- All three require definedness checking.
- These laws and definedness checking are implemented in the IMPS system.

Substitution for a Variable

- In FOL the law of universal instantiation is:

$$(\forall x \,.\, A) \Rightarrow A[x \mapsto t] \quad \text{where } t \text{ is free for } x \text{ in } A.$$

- In FOLwU is is:

$$((\forall x \,.\, A) \wedge t\downarrow) \Rightarrow A[x \mapsto t] \quad \text{where } t \text{ is free for } x \text{ in } A.$$

- Thus, in a logic with undefinedness, a substitution is a mapping of variables to defined terms.
- The substitutions found by naive matching may not be legitimate — the definedness of the target terms must checked!
- Candidate substitutions can be used before they are checked!
  - See the IMPS method of finding applicable macetes.

Equality Substitution

- In FOL the law of equality substitution is:

  $s = t \Rightarrow (A \Rightarrow A^*)$ where A^* is the result of replacing one occurrence of s in A with t, provided that the occurrence of s is not a variable immediately after $\forall$.

- In FOLwU it is:

  $s \simeq t \Rightarrow (A \Rightarrow A^*)$ where A^* is the result of replacing one occurrence of s in A with t, provided that the occurrence of s is not a variable immediately after $\forall$.

- Thus, in a logic with undefinedness, term rewriting is based on quasi-equality instead of equality.

Definition of a Constant

- A definition of a constant c is an axiom of the form $c = t$.
- In a traditional logic, the following syntactic conditions must be verified before the definition can be added to a theory T:
  - c is new, i.e., it is not in the language of T.
  - t does not contain c.
- In a logic with definedness, a definedness condition must also be verified:
  - $t\!\downarrow$ is valid in T.
- Let T be a FOLwU theory of the real numbers.
  - $2 = (\mathrm{I}\, x \,.\, x = 1 + 1)$ is a legitimate definition in T.
  - inv-zero $= 0^{-1}$ is not a legitimate definition in T.

Definedness Checking

- Effective definedness checking is crucial for an implementation of a logic with undefinedness.
  - Reasoning leads to a proliferation of definedness conditions, most of which need to be checked.
  - Definedness checking is an undecidable problem.
- The IMPS simplifier can automatically check almost all definedness conditions that typically arise in IMPS proofs.
  - When the IMPS simplifier fails, a nontrivial argument is usually needed to verify the definedness condition.

Definedness Checking in IMPS

- The IMPS simplifier checks definedness in a type, i.e, formulas of the form $t \downarrow \alpha$.
- The simplifier employs theorems concerning the domain and range of functions and the relationship between types such as:
  - "f is total".
  - $C \Rightarrow \alpha \ll \beta$.
  - $\forall\, x_1 : \alpha_1, \ldots, x_n : \alpha_n \,.\, C(x_1, \ldots, x_n) \Rightarrow (f(x_1, \ldots, x_n) \downarrow \alpha)$.
  - $\forall\, x_1 : \alpha_1, \ldots, x_n : \alpha_n \,.\, C(x_1, \ldots, x_n, f(x_1, \ldots, x_n))$
- The simplifier also uses the method of local contexts and cached results.
- Example:

$$\forall x, y : \mathbf{Z}, z : \mathbf{Q} \,.\, 2 < z \Rightarrow ((x * y - 3!/|z|) \downarrow \mathbf{Q}).$$

- Definedness checking in IMPS is very effective in practice!

Conclusion

- A practical logic needs to handle undefinedness effectively.
- The traditional approach to undefinedness, which is entrenched in mathematical practice, is very effective.
- This approach can be formalized in a traditional logic by slightly modifying the logic.
  - The syntax needs little, if any, modification.
  - The semantics is changed to admit partial functions and undefined terms.
  - The proof system needs new laws about definedness and modified laws involving variable and equality substitution.
- A logic of this kind can be effectively implemented.
  - Good definedness checking is crucial.

References I

[1] 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.

[2] T. Burge.

Truth and Some Referential Devices.

PhD thesis, Princeton University, 1971.

[3] W. M. Farmer.

A partial functions version of Church's simple theory of types.

Journal of Symbolic Logic, 55:1269–91, 1990.

References II

[4] W. M. Farmer.

A simple type theory with partial functions and subtypes.

Annals of Pure and Applied Logic, 64:211–240, 1993.

[5] 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.

[6] W. M. Farmer.

Reasoning about partial functions with the aid of a computer.

Erkenntnis, 43:279–294, 1995.

References III

[7] W. M. Farmer.

STMM: A Set Theory for Mechanized Mathematics.

Journal of Automated Reasoning, 26:269–289, 2001.

[8] W. M. Farmer.

Formalizing undefinedness arising in calculus.

In D. Basin and M. Rusinowitch, editors, *Automated Reasoning—IJCAR 2004*, volume 3097 of *Lecture Notes in Computer Science*, pages 475–489. Springer-Verlag, 2004.

References IV

[9] W. M. Farmer.

Chiron: A multi-paradigm logic.

In R. Matuszewski and A. Zalewska, editors, *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar and Rhetoric*, pages 1–19. University of Białystok, 2007.

[10] W. M. Farmer.

Chiron: A set theory with types, undefinedness, quotation, and evaluation.

SQRL Report No. 38, McMaster University, 2007.

Revised 2009.

References V

[11] W. M. Farmer.

Andrews' type system with undefinedness.

In C. Benzmüller, C. Brown, J. Siekmann, and R. Statman, editors, *Reasoning in Simple Type Theory: Festschrift in Honor of Peter B. Andrews on his 70th Birthday*, Studies in Logic, pages 223–242. College Publications, 2008.

[12] W. M. Farmer.

The seven virtues of simple type theory.

Journal of Applied Logic, 6:267–286, 2008.

DOI:10.1016/j.jal.2007.11.001.

References VI

[13] W. M. Farmer and J. D. Guttman.

A set theory with support for partial functions.

Studia Logica, 66:59–78, 2000.

[14] W. M. Farmer, J. D. Guttman, and F. J. Thayer.

IMPS: An Interactive Mathematical Proof System.

Journal of Automated Reasoning, 11:213–248, 1993.

[15] C. B. Jones.

References relating to reasoning about partial functions, 2007.

Available at `http://homepages.cs.ncl.ac.uk/cliff.jones/ftp-stuff/lpf-refns.pdf`.

References VII

[16] L. G. Monk.

PDLM: A Proof Development Language for Mathematics.

Technical Report M86-37, The MITRE Corporation, Bedford, Massachusetts, 1986.

[17] O. Müller and K. Slind.

Treating partiality in a logic of total functions.

The Computer Journal, 40:640–652, 1997.

[18] D. L. Parnas.

Predicate logic for software engineering.

IEEE Transactions on Software Engineering, 19:856–861, 1993.

References VIII

[19] R. Schock.

Logics without Existence Assumptions.

Almqvist & Wiksells, Stockholm, Sweden, 1968.