A Description Operator for First-Order-Logic

Timo Lücke

Mathematics
Jacobs University Bremen
Campus Ring 1
28759 Bremen
Germany

Type: *Bachelor's Thesis*
Date: *August 23, 2014*
Advisors: *Prof. Dr. Michael Kohlhase, Dr. Florian Rabe*
Second Reader: *Prof. Dr. Stefan Maubach*

Abstract

First-order-logic (FOL) is immensely important in wide areas of mathematics and computer science. There is a wide array of extensions by minor features which make FOL more expressive.

One such feature is a description operator which we look at in more detail in this thesis. It is surprising that there is no logic which formally extends FOL with a description operator. is essential for formalizing most areas of Mathematics. In this thesis we introduce a new logic FOL$^{\delta}$, which provides a description operator. We then go on to prove soundenss and completeness and give a semantics-preserving translation into FOL

We utilize the Edinburgh Logical Framework (LF) to define our new logic. One particular difficulty of the description operator is that it is a partial operator, as it is only defined if the predicate is uniquely satisfiable. We capture this partiality by making proofs part of terms. This allows terms to carry witnesses of proofs of their definedness.

Contents

1 Introduction and Related Work

For a lot of applications of logic both in mathematics and computer science first-order logic (FOL) is the logic of choice, as it is, in some sense, the most expressive logic for which there is a complete calculus. However, this comes at the expense of not being able to express certain mathematical concepts (like partial functions) elegantly.

There are possible ways to improve this situation. One possibility is to move to higher-order logics. However, by doing this we lose a number of desired theoretical properties (most importantly the existence of a complete calculus with respect to standard set theoretical semantics) and the fairly large amount

of computer support that has been developed for FOL. Therefore, a widely studied approach is to adapt FOL.

Many extensions have been extremely well-studied, but -somewhat surprisingly- the description operator, which is prevalent in informal mathematics, has received relatively little attention: Phrases like "let x be the unique element such that $P(x)$ holds" are very widespread in mathematical textbooks and papers alike, but cannot be formalized directly in pure FOL.

In order to formalize such phrases we extend FOL with a definite description operator δ, an operator that for any provable sentence of the form $\exists!x.P(x)$ allows us to obtain the unique witness x.

We present a new approach to the formal definition of this operator. Additionally we provide a semantics-preserving translation procedure of our new logic into FOL. Translations between different first-order logics have been given in [7] and [9]

Related Work There are many approaches to this, for example CASL[7] in the field of algebraic specification which extends FOL by a number of orthogonal features (sorts, subsorts and partial functions among others) or TPTP[6] which is a concrete syntax mostly used in automated reasoning and includes sorts, interpreted arithmetics and shallow polymorphisms. Another possible addition to FOL is dependent typing[8] (the type of later arguments expected by a function may be determined by its earlier arguments).

As noted above, some extensions also include partial functions. Since the description operator is partial, it is worth looking at the different approaches to deal with partiality. In most cases they accept well-formed undefined terms and introduce either a special constant $\bot$ (as in [11]) for undefined or a predicate def (as in [7]), they deal with undefinedness at the semantic level. We go a different route and avoid undefined terms by making them ill-formed. This has the drawback that it usually leads to definedness of terms being undecidable. To overcome this issue we can (as done in this thesis) require terms to carry a proof of their definedness (using reasoning support to help with the high number of proof obligations put on the author). This leads to the problem of representing proofs inside terms.[1]

A different approach is widespread in higher-order-logic (HOL): Church's original paper [10] already included a choice operator (of which the description operator is a simple special case). Its approach is somewhat similar to the $\bot$ mentioned above (but without a special constant being introduced): the axioms guarantee that every type is non-empty, and the choice operator (of which the description operator is a special case) is defined as an operator from formulas with one free variable to terms. The disadvantage of this approach is [2].

Methods We will use the Edinburgh Logical Framework (LF) [5] to represent our logic. This was already done for the extension of FOL by dependent types

[1] citations for partial fun, HOL

[2] non-empty axiom, no explicit value?, counterintuitive?

in [8]. It allows an elegant and natural formalization of the description operator and its semantics by incorporating proofs in terms, resulting in a logic we call FOL$^\delta$. More specifically, our operator takes two arguments, a formula $A = \exists!x.F(x)$ and a proof P of A, and returns the unique witness.

Outline In Section 2 we first present standard FOL and then give an introduction into LF, using FOL as an example. We then discuss the extensions necessary to incorporate the description operator in Section 3. We give a formalization of basic concepts of set theory as a case study in Section 4. We prove soundness and completeness in Sect. 3 and 5 respectively. Our completeness result is derived from a semantics-preserving translation, which we also give in Section 5, before giving a brief conclusion in Section 6.

All LF-encodings are available at `https://svn.kwarc.info/repos/supervision/gr/2014/luecke_timo/project/`.[3]

2 Preliminaries

2.1 First-Order-Logic

In this section, we introduce first-order logic in order to give an overview of the definitions and notations we will use. These definitions are taken from [1] and expanded where necessary.

Definition 1 (Signatures)**.** A FOL-*signature* is a triple (Σ_f, Σ_p, ar) where Σ_f and Σ_p are disjoint sets of function and predicate symbols, respectively, and $ar : \Sigma_f \cup \Sigma_p \to \mathbb{N}$ assigns arities to symbols. We will treat constants and boolean variables as the special case of arity 0.

Definition 2 (Expressions)**.** A FOL-*context* is a list of variables. For a signature Σ and a context Γ, the *terms* over Σ and Γ are formed from the variables in Γ and the application of function symbols $f \in \Sigma_f$ to terms. The *formulas* over Σ and Γ are formed by the application of predicate symbols $p \in \Sigma_p$ to terms as well as through application of connectives to formulas. In addition we define $\exists!x.P(x) := \exists z.P(x) \wedge (\forall y.(P(y) \Rightarrow y \doteq x))$.

$$
\begin{array}{lll}
term & ::= & x \\
 & & |f(\underbrace{term, ...term}_{ar(f)}) \mid \\
form & ::= & \neg form \mid form \wedge form \mid form \vee form \\
 & & |form \Rightarrow form \mid term \doteq term \\
 & & |true \mid false \\
 & & \mid \forall x.form(x) \mid \exists x.form(x) \\
 & & p(\underbrace{term, term, ...term}_{ar(p)})|
\end{array}
$$

Formulas in the empty context are called Σ-*sentences*, and we write $\mathbf{Sen}(\Sigma)$ for the set of sentences.

[3] so far they aren't

Definition 3 (Theories)**.** A FOL-*theory* is a pair (Σ, Θ) for a signature Σ and a set $\Theta \subseteq \mathbf{Sen}(\Sigma)$ of *axioms*.

Definition 4 (Signature Morphisms)**.** Given two signatures $\Sigma = (\Sigma_f, \Sigma_p, ar)$ and $\Sigma' = (\Sigma'_f, \Sigma'_p, ar')$, a FOL-*signature morphism* $\sigma : \Sigma \to \Sigma'$ is an arity-preserving mapping from Σ_f to Σ'_f and from Σ_p to Σ'_p.

The *homomorphic extension* of σ – which we also denote by σ – is the mapping from terms and formulas over Σ to terms and formulas over Σ' that replaces every symbol $s \in \Sigma_f \cup \Sigma_p$ with $\sigma(s)$. The *sentence translation* $\mathbf{Sen}(\sigma) : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$ arises as the special case of applying σ to sentences.

Example 5 (Monoids and Groups). We will use the theories $Monoid = (MonSig, MonAx)$ and $Group = (GrpSig, GrpAx)$ of monoids and groups as running examples. $MonSig_f$ is the set $\{\circ, e\}$ where $\circ$ is binary (written infix) and e is nullary, and $MonSig_p$ is empty. $MonAx$ consists of the axioms for

- associativity: $\forall x\ \forall y\ \forall z\ x \circ (y \circ z) \doteq (x \circ y) \circ z$,
- left-neutrality: $\forall x\ e \circ x \doteq x$,
- right-neutrality: $\forall x\ x \circ e \doteq x$.

The theory *Group* extends *Monoid*, i.e., *GrpSig* adds a unary function symbol *inv* (written as superscript $^{-1}$) to *MonSig*, and *GrpAx* adds axioms to *MonAx* for the following:

- left-inverseness: $\forall x\ x^{-1} \circ x \doteq e$,
- right-inverseness: $\forall x\ x \circ x^{-1} \doteq e$.

The inclusion mapping *MonGrp* is a signature morphism from *MonSig* to *GrpSig*.

There are various ways to define the *proof theory* of FOL. Here we choose the natural deduction calculus (ND) with introduction and elimination rules.

Definition 6 (Proof Theoretical Theorems)**.** Given a theory (Σ, Θ), we say that $F \in \mathbf{Sen}(\Sigma)$ is a *proof theoretical theorem* of (Σ, Θ) if the judgment $F_1, \ldots, F_n \vdash_\Sigma F$ is derivable for some $\{F_1, \ldots, F_n\} \subseteq \Theta$ using the calculus shown in Fig. **??**. We write this as $\Theta \vdash_\Sigma F$.

Lemma 7. *The 2 rules at the bottom of* **??** *are derivable from the others.*

Proof. Straightforward □

Definition 8 (Proof Theoretical Theory Morphisms)**.** A signature morphism from Σ to Σ' is a *proof theoretical theory morphism* from (Σ, Θ) to (Σ', Θ'), written $\sigma : (\Sigma, \Theta) \xrightarrow{\mathsf{P}} (\Sigma', \Theta')$, if $\mathbf{Sen}(\sigma)$ maps the axioms of (Σ, Θ) to proof theoretical theorems of (Σ', Θ'), i.e., for all $F \in \Theta$, $\Theta' \vdash_{\Sigma'} \mathbf{Sen}(\sigma)(F)$ holds.

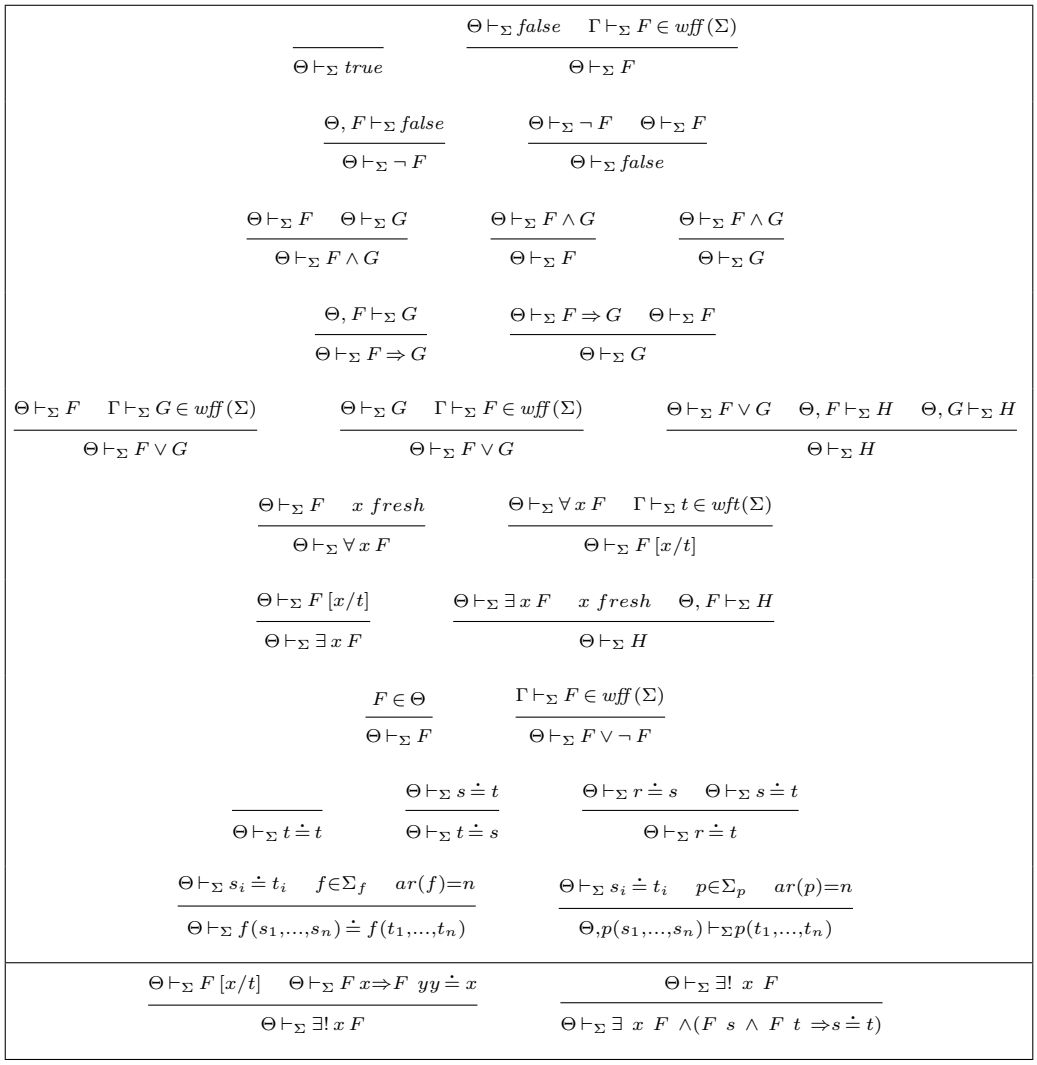

Figure 1: Proof Rules (derived rules at the bottom)

Lemma 9 (Proof Translation)**.** *Assume a proof theoretical theory morphism* $\sigma : (\Sigma, \Theta) \to (\Sigma', \Theta')$*. If* F *is a proof theoretical theorem of* (Σ, Θ)*, then* $\mathbf{Sen}(\sigma)(F)$ *is a proof theoretical theorem of* (Σ', Θ')*. In other words, provability is preserved along proof theoretical theory morphisms.*

We now go on to develop the *model theory* of FOL.

Definition 10 (Models of a FOL-Signature)**.** A FOL-*model of a signature* Σ is a pair (U, I) where U is a non-empty set (called the *universe*) and I is a function

of Σ-symbols (called the interpretation) such that

- $f^I \in U^{(U^n)}$ for $f \in \Sigma_f$ with $ar(f) = n$,
- $p^I \subseteq U^n$ for $p \in \Sigma_p$ with $ar(p) = n$.

We write $\mathbf{Mod}(\Sigma)$ for the class of Σ-models.

Definition 11 (Model Theoretical Semantics)**.** Assume a signature Σ, a context Γ, and a Σ-model $M = (U, I)$. An *assignment* is a mapping from Γ to U. For an assignment α, the *interpretation* $I^\alpha(t) \in U$ of terms t and $I^\alpha(F) \in \{0, 1\}$ of formulas F over Σ and Γ are defined in the usual way by induction on the syntax:

- $I^\alpha(x) \ = \alpha(x)$ for $x \in \Gamma$
- $I^\alpha(f(t_1 \dots t_n)) = f^I(I^\alpha(t_1) \dots I^\alpha(t_n))$
- $I^\alpha(p(t_1 \dots t_n)) = 1$ iff $(I^\alpha(t_1) \dots I^\alpha(t_n)) \in p^I$
- $I^\alpha(F \wedge G) = min(I^\alpha(F), I^\alpha(G))$
- $I^\alpha(F \vee G) = max(I^\alpha(F), I^\alpha(G))$
- $I^\alpha(\neg F) = 1$ iff $I^\alpha(F) = 0$
- $I^\alpha(F \Rightarrow G) = 1$ iff $I^\alpha(G) = 1$ or $I^\alpha(\neg F) = 1$
- $I^\alpha(s \doteq t) = 1$ iff $I^\alpha(s) = I^\alpha(t)$
- $I^\alpha(\forall x\ F) = 1$ iff forall $u \in U \quad I^{\alpha,[u/x]}(F) = 1$
- $I^\alpha(\exists x\ F) = 1$ iff exists $u \in U \quad I^{\alpha,[u/x]}(F) = 1$

Given a sentence F, we write $M \models_\Sigma F$ if $[\![F]\!]^M = 1$.
Given a theory (Σ, Θ), we write the class of (Σ, Θ)-models as

$$\mathbf{Mod}(\Sigma, \Theta) = \{M \in \mathbf{Mod}(\Sigma) \mid M \models_\Sigma F \text{ for all } F \in \Theta\}.$$

Since we are especially interested in unique existence, but it is not a primary object in the logic, we provide its interpretation here.

Lemma 12.
$I^\alpha(\exists!x\ F) = 1$ *iff there exists a unique* $u \in U \quad I^{\alpha,[u/x]}(F) = 1$

Proof. Straightforward □

Definition 13 (Model Theoretical Theorems)**.** Given a theory (Σ, Θ), we say that $F \in \mathbf{Sen}(\Sigma)$ is a *model theoretical theorem* of (Σ, Θ) if the following holds for all Σ-models M: If $M \models_\Sigma A$ for all $A \in \Theta$, then also $M \models_\Sigma F$. We write this as $\Theta \models_\Sigma F$.

Definition 14 (Model Reduction)**.** Given a signature morphism $\sigma : \Sigma \to \Sigma'$ and a Σ'-model $M' = (U, I')$, we obtain a Σ-model (U, I), called the *model reduct* of M' along σ, by putting $s^I = \sigma(s)^{I'}$ for all symbols of Σ. We write $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ for the induced model reduction.

Definition 15 (Model Theoretical Theory Morphisms)**.** Given two theories (Σ, Θ) and (Σ', Θ'), a *model theoretical theory morphism* from (Σ, Θ) to (Σ', Θ'), written $\sigma : (\Sigma, \Theta) \xrightarrow{\mathtt{M}} (\Sigma', \Theta')$, is a signature morphism from Σ to Σ' such that $\mathbf{Mod}(\sigma)$ reduces models of (Σ', Θ') to models of (Σ, Θ), i.e, for all $M' \in \mathbf{Mod}(\Sigma', \Theta')$, we have $\mathbf{Mod}(\sigma)(M') \in \mathbf{Mod}(\Sigma, \Theta)$.

Example 16 (Continued). The integers form a model $Int = (\mathbb{Z}, +, 0, -)$ for the theory of groups (where we use a tuple notation to give the universe and the interpretations of $\circ$, e, and inv, respectively). The model reduction $\mathbf{Mod}(MonGrp)(Int) = (\mathbb{Z}, +, 0)$ along $MonGrp$ yields the integers seen as a model of the theory of monoids.

We have given both proof theoretical and model theoretical definitions of *theorem* and *theory morphism*. In general, these must be distinguished to avoid a bias towards proof or model theory. However, they coincide if a logic is sound and complete:

Theorem 17 (Soundness and Completeness)**.** *Assume a FOL-theory* (Σ, Θ) *and a* Σ*-sentence* F*. Then* $\Theta \vdash_\Sigma F$ *iff* $\Theta \models_\Sigma F$*. Therefore, for a FOL-signature morphism* $\sigma : \Sigma \to \Sigma'$*, we have* $\sigma : (\Sigma, \Theta) \xrightarrow{\mathtt{P}} (\Sigma', \Theta')$ *iff* $\sigma : (\Sigma, \Theta) \xrightarrow{\mathtt{M}} (\Sigma', \Theta')$*.*

2.2 The Edinburgh Logical Framework

In this section we introduce the Edinburgh Logical Framework[5]. We again follow [1] in large parts, but use a different running example.

LF ([5]) is a dependent type theory that extends simple type theory with dependent function types. The main use of LF is as a *logical framework* in which deductive systems are represented.

We develop the syntax and semantics of LF along with an example representation of the first-order logic defined above. Typically, *kinded type families* are declared to represent the syntactic classes of the encoded logic. For FOL, we declare

$$\begin{array}{lcl} form & : & \mathtt{type} \\ term & : & \mathtt{type} \\ ded & : & form \to \mathtt{type} \\ contra & : & type = \{\mathrm{a}\}\ \mathrm{ded\ a} \end{array}$$

Here `type` is the LF-kind of types, and $form$ is an LF-type whose LF-terms represent the FOL-formulas. $ded : form \to \mathtt{type}$ is the kind of types that are proofs of formulas: for any formula A $ded\ A$ is the type of proofs of A.

Typed constants are declared to represent the constructors of the expressions of the represented system. For the syntax of FOL, we declare

eq	:	$term \to term \to form$	$1 = 2$ prec 25
$true$	:	$form$	
$false$	:	$form$	
and	:	$form \to form \to form$	$1 \wedge 2$ prec 15
or	:	$form \to form \to form$	$1 \vee 2$ prec 15
$impl$	:	$form \to form \to form$	$1 \Rightarrow$ prec 10
neg	:	$form \to form$	$\neg 1$ prec 20
$forall$	:	$(term \to form) \to form$	$\forall 1$ prec 30
$exists$	:	$(term \to form) \to form$	$\exists 1$ prec 30
$existsU$	:	$(term \to form) \to form$	$\exists! 1$ prec 30

This is simply the syntax given in the Sect. 2.1 expressed in LF. Note that $existsU$ can be defined to be an abbreviation. The second part of each definition gives the notation that we will use and the precedence of the argument for bracket elision.

LF employs the Curry-Howard correspondence to represent proofs-as-terms [4] and extends it to the *judgments-as-types* methodology [3]. To exemplify this we add to the syntax of FOL the proof rules of the natural deduction calculus for $\wedge$ and $\forall$, the full encoding is availabe at `https://svn.kwarc.info/repos/supervision/gr/2014/luecke_timo/project/FOL`

$trueI$	:	$ded\ true$
$andI$	:	$\{A, B\}\ ded\ A \to ded\ B \to ded\ A \wedge B$
$andEl$	:	$\{A, B\}\ ded\ A \wedge B \to ded\ A$
$andEr$	:	$\{A, B\}\ ded\ A \wedge B \to ded\ B$
$forallI$	:	$\{A\}(\{x : term\}\ ded\ (A\ x)) \to ded\ (\forall[x]\ A\ x)$
$forallE$	:	$\{A, t : term\} ded\ (\forall[x : term]\ A\ x) \to ded\ (A\ t)$

Consequently, all inference rules are represented as appropriately typed constants, and inference rules as operators on the family of types of proofs.

Note that these operators like $andI$ really take 4 arguments. This uses the main feature of dependent type theory: The first two arguments A and B may occur in the types of the later arguments and in the return type. Implementations, like the MMT implementation [2] that we use, treat A and B as *implicit arguments* and infers their values from the types of the other arguments. Thus, we can write (for proofs P of A and Q of B) $andI\ P\ Q$ instead of $andI\ A\ B\ P\ Q$.

3 A Description Operator for FOL

We extend FOL with the aforementioned δ-operator. It is a term constructor that allows us to obtain from a unary predicate $P(x)$ the unique witness x such that $P(x)$. However, simply defining δ as an operator from predicates to terms only gives us a partial operator, since it it is only defined on any predicate F for which $\exists! x.F(x)$ is provable. To remedy this situation our δ takes as a second argument a proof P of the fact $\exists! x\ F(x)$. Since $ded(\exists! x\ F(x))$ is a type, we obtain a well-typed total operator by making use of dependent type theory.

3.1 Syntax and Proof theory

We include the operator into our syntax by extending our above definition of expressions to (additions underlined):

Definition 18 (Expressions)**.**

$$
\begin{array}{lll}
term & ::= & x \mid f(\underbrace{term, ...term}_{ar(f)}) \\
 & & \mid \underline{\delta(form, proof)} \\
form & ::= & \neg form \mid form \wedge form \mid form \vee form \\
 & & \mid form \Rightarrow form \mid term \doteq term \\
 & & \mid true \mid false \\
 & & \mid \forall x.form \mid \exists x.form \\
 & & p(\underbrace{term, ...term}_{ar(p)}) \mid \\
proof & ::= & \underline{deltaaxiom} \mid ...
\end{array}
$$

The dots in the *proof* part of the grammar represent exactly the rules from **??**. *deltaaxiom* is the following rule:

$$\frac{}{\Theta \vdash_\Sigma F(\delta(F,P))}$$

[4]

Note that we now include proofs in terms. This breaks with the standard hierarchy widespread in logics which is that formulas contain terms but not proofs. However, in LF there is no need for this distinction since proofs themselves are represented as terms, so we can include proofs directly into the grammar of our logic. The two additions to our grammar above can be included into the encoding with the following two declarations. yielding the following additions:

$$
\begin{array}{lll}
\delta & : & \{A : term \to form\}\ (ded\ \exists!\ A) \to term \quad \delta\ 1\ 2 \text{ prec } 25 \\
deltaax & : & \{A, P : ded\ (\exists! A)\}\ ded\ (A\ (delta\ A\ P))
\end{array}
$$

We pick up our example from abstract algebra again. Even though the description operator is not strictly necessary, it helps streamline many of the definitions. Below we provide an LF-encoding of basic theories of monoids and groups.

Example 19 (Algebra (continued from Ex. 5).

extends FOLd with

$$
\begin{array}{lll}
* & : & term \to term \to term \\
assoc & : & ded\ \forall x.\forall y.\forall z.\ (x * y) * z \doteq x * (y * z) \\
unitexists & : & ded\ \exists e.\forall x.\ e * x \doteq x \wedge x * e \doteq x \\
unitunique & : & ded\ \exists! e\forall x.\ e * x \doteq x \wedge x * e \doteq x = \ldots \\
e & : & \delta\ ([x]\forall y.x * y \doteq y \wedge y * x \doteq y)\ unitunique
\end{array}
$$

[4] already was begincenter, needs to be textmode

This provides the definition of a monoid (*unitunique* is a proof for the uniqueness of the unit in a monoid, given at [5]).

Further, we obtain the theory group by extending the above with the following declarations:

invexists	:	$\forall x.\exists y.\ x * y \doteq e$
invunique	:	ded $\{x\}\exists!y.\ x * y \doteq e$
inv	:	$term \to term \quad = \{x\}\ \delta([y]\ y * x \doteq e)\ unqinv$

This adds the inverses to the monoid definiton to obtain groups.

Note that in both cases (unit and inverse) we did not have to define the objects as primitive symbols, but obtained it using the existence axiom. This is a recurring theme, as in a lot of mathematical theories we postulate the existence of an object, and prove uniqueness from its properties. In these situations, the use of the δ-operator is very natural.

We need to show that introducing proofs into terms does not affect the semantics.

Theorem 20 (Proof Irrelevance)**.** *Given a fomula F and proofs P, P' of $\exists!x.F(x)$ $\delta(F,P) \doteq \delta(F,P')$.*

Proof. The proof for this is straightforward: by our axiom-scheme we have $F(\delta(F,P)) \wedge F(\delta(F,P'))$. P is already a proof for $\exists!x.F(x)$ which is equivalent to $\exists x.F(x) \wedge (F(x) \wedge F(t) \Rightarrow x \doteq t)$. Implication elimination on the second part of this, using the conjunction above gives us exactly the desired result. □

The LF encoding of this proof can be found at `https://svn.kwarc.info/repos/supervision/gr/2014/luecke_timo/project/FOLd` [6].

3.2 Model Theory

The only addition to the model theory is the interpretation of $\delta(F,P)$:

- $I^\alpha(\delta(F,P)) = u$, where $u \in U$ s.t.$I^{\alpha,[u/x]}(F) = 1$.

Theorem 21. *The interpretation function is well-defined and the resulting logic is sound.*

Proof. We prove these two statements by a mutual induction over the structure of expressions:
For both statements, the cases that do not include δ-terms are just like in the proofs for the corresponding statements in standard FOL, which are well studied. These serve as induction hypotheses. For the well-definedness of $I^\alpha(\delta(F,P))$ the IH for soundness yields that there is exactly one $u \in U$ s.t.$I^{\alpha,[u/x]}(F) = 1$. For the soundness of the proof rule we have to show that $I^\alpha(F(\delta(F,P))) = 1$. This holds due to the interpretation of δ. □

[5] add link to encoding
[6] not yet

4 A Formalization of Set Thery

In this section we present a formalization of the basic concepts of set theory, focusing on the definitions for which the description operator is relevant. We leave out some syntax that in LF serves to support the parser, but would only hurt readability here.

theory Sets : http://cds.omdoc.org/urtheories?LF =
include ?FOLd
implchain : A,B,C ded (A ⇒ B) → ded(B ⇒ C) → ded(A ⇒C)
= [A,B,C][p: ded A ⇒ B][q: ded B ⇒ C] impI [x: ded A] impE q (impE p x)

equivchain: A,B,C ded (A ⇔ B) → ded(B ⇔ C) → ded(A ↔ C)
= [A,B,C][p: ded A ⇔ B, q: ded B ⇔C]
equivI ([x: ded A] equivEl q (equivEl p x))
([y: ded C] equivEr p (equivEr q y))

equivsym: A,B ded (A ⇔ B) → ded (B ⇔ A)
= [A,B][p]
equivI ([x] equivEr p x)
([x] equivEl p x)

These are some abbreviations to make the rest of the encoding better readable. They signify, in order, transitivity of implication and transitivity+symmetry of equivalence.
in : i → i → form # 1 ∈ 2
subset: i → i → form = [A,B] forall ([x] ((x âĂIJâĹĹ A) ⇒ (x âĂIJâĹĹ B)))

extensionality : x,y ded (forall [z] (in z x) ⇔ (in z y)) âĘŠ ded x == y

isempty : i → form = [x] Âň exists [y] in y x
emptyAx : ded exists [x] isempty x
empty : i= delta ([x] isempty x)
(exUI emptyAx
([A,B] [P: ded isempty A, Q: ded isempty B] faI ([x] a)))

ispowerset: i → i → form = [A,B] forall [x] ((subset x A) ⇔ (in x B))
powersetAx: ded forall [A] exists [B] ispowerset A B
powerset: i → i = [A] delta ([P] ispowerset A P)
(exUI (faE powersetAx A)
([U,V] [Q: ded ispowerset A U, R: ded ispowerset A V]
extensionality (faI [x] (equivchain (equivsym (faE Q x)) (faE R x)))))

isunion: i → i → form= [A, U] forall [x] ((exists ([a] (a âĂIJâĹĹ A) âĹğ (x âĂIJâĹĹ a))) ⇔ in x
U)

unionAx: ded forall [A] exists [U] (isunion A U)
union: i → i =
[A] delta ([U] isunion A U)
(exUI (faE unionAx A) (
[U,V, P: ded isunion A U, Q: ded isunion A V]
extensionality (faI [x] (equivchain (equivsym (faE P x)) (faE Q x)))))

comprehensionAx: A,F:i → form ded exists [C] forall [x] ((x âĂIJâĹĹ C) ⇔ ((x âĂIJâĹĹ A) âĹğ (F x)))

ispairof: i → i → i → form= [a,b,J] forall [z] (z == a âĹĺ z == b) ⇔ (in z J)
unpairAx: ded forall[x] forall[y] exists [A] ispairof x y A
unpair: i → i → i= [a,b] delta ([J] ispairof a b J)
(exUI (faE(faE unpairAx a) b) (
[J,K, P: ded ispairof a b J, Q: ded ispairof a b K]
extensionality (faI [x] (equivchain (equivsym (faE P x)) (faE Q x)))))

binunion : i → i → i = [x][y] union (unpair x y)# 1 âĹł 2
singleton : i → i = [x] unpair x x

There are a couple of things to note here. First of all, notice that we need significantly less primary definitions than we would if we would be working without the description operator. empty, powerset, union, and unpair would all have to be primary definitions otherwise. Additionally, our development mirrors mathematical practice by showing uniqueness of the objects out of the properties, which is especially prevalent in many introductions to set theory. Also note that, after defining abbreviations for the properties, the proofs look very similar to each other. This might be helpful in developing tool support for these proofs in the future.

5 Meta-Logical Properties

It is well-known that a description operator can in principle be eliminated by translating into pure FOL. We spell out such a translation for our formalization now. This also allows us to recover completeness w.r.t. the standard set-theoretical semantics. It is additionally motivated by the large amount of tool support that has been developed for FOL over the last decades.

The basic idea for the translation is first replacing the δ-terms with corresponding unique existence clauses that bind variables to the values of the δ-terms and then replacing these in the formula.

We first need an auxiliary definition:

Definition 22 (simple δ-terms)**.**
We call t a *simple δ-term* if it is of the form $\delta(F, P)$ and F contains no δ-terms.

Definition 23. For any signature Σ we define a mapping σ_Σ from $FOL^\delta\Sigma$ formulas to FOL_Σ[7] formulas. The definition proceeds by induction over formulas. The only non-trivial case is the one for atomic formulas $F = p(t_1, ..., t_n)$, where we distinguish two cases:

- If all occurring δ-terms are simple, we define, assuming p contains the δ-terms $\delta(F_1, Q_1) \ldots \delta(F_k, Q_k)$

  $\sigma(p(t_1 \ldots t_n)) := \exists!x_1.F_1(x_1) \wedge \exists!x_2.F_2(x_2) \ldots \exists!x_k.F_k(x_k) \wedge [x_i/\delta(F_i, P_i)]F(t_1 \ldots t_n)$

- If p contains δ-terms which aren't simple, we define

  $$\sigma(F) = \sigma(F'),$$

  where F' is obtained by replacing all occurrences of δ-terms $\delta(F_i, P_i)$ by $\delta(\sigma(F_i), P_i)$

We then define $\sigma(A \wedge B) := \sigma(A) \wedge \sigma(B)$ and correspondingly for the other connectives.

Example 24 (continued from ??).
Consider the following sentence from our theory of monoids: $\sigma(x * e \doteq x)$
$= \sigma(x * \delta\ ([x]\forall y.x * y \doteq y \wedge y * x \doteq y)\ (unitunique) \doteq x$
$= \exists!t.(\forall y.t * y \doteq y \wedge y * t \doteq y) \wedge x * t \doteq x$
This sentence is now in pure FOL and semantically equivalent to the original sentence.[8]
For a second example containing nested δ-terms, we consider a formula from the theory of groups:
$\sigma(inv\ x \doteq y)$
$= \sigma(\ (\delta\ ([b]\ b * x \doteq (\delta\ ([a]\forall[z]a * z \doteq z)\ P) \doteq e)\ Q) \doteq y)$
$= \sigma(\ (\delta\ ([b]\sigma(\ b * x \doteq (\delta\ ([a]\forall[z]a * z \doteq z)\ P)) \doteq e)\ Q) \doteq y)$
$= \sigma(\ (\delta\ (\exists!t.(\forall z.t * z \doteq z) \wedge b * x \doteq t) \doteq e)\ Q') \doteq y)$
$= \exists!s.(\exists!t.(\forall z.t * z \doteq z) \wedge s * x \doteq t) \doteq e) \wedge s \doteq e$[9]

Theorem 25. *For any* Σ, $\vdash_\Sigma^{FOL^\delta} A \iff \sigma(A)$

Proof. The general statement follows immediately if we prove it for atomic formulas. More specifically, we only need to consider the case without nested δ-terms, since the nested-δ case would follow (the translation on the δ-free formulas at the bottom of any nesting are semantics-preserving and turn the translations of the next level into one of δ-free atomic formulas).

So, using the situation from Def. 22 we have to show:
$F(t_1 \ldots t_n) \iff \exists!x_1.F_1(x_1) \wedge \exists!x_2.F_2(x_2) \ldots \exists!x_k.F_k(x_k) \wedge [x_i/\delta(F_i, P_i)]F(t_1 \ldots t_n)$

[7] is it really the same sigma?
[8] work out inv inv x ==e
[9] check

The unique existence clauses are equivalent to the well-formedness of the δ-terms. Additionally, we can easily show that $x_i = \delta(F_i, Q_j)$, since we know $F_i(x_i)$ and x_i is unique, and together with the δax this gives us equality. This implies that, given the unique existence clauses, $F(t_1 \dots t_n) \iff [x_i/\delta(F_i, P_i)]F(t_1 \dots t_n)$. □

Theorem 26 (Model Theoretical Preservation)**.** *For any assignment $I^\alpha(F) = 1$ in FOL$^\delta$ iff $I^\alpha(\sigma(F)) = 1$ in FOL.*

Proof. From our theorem above we can conclude that $I^\alpha(A) = 1$ in FOL$^\delta$ iff $I^\alpha(\sigma(A)) = 1$ in FOL$^\delta$. However, since $\sigma(A)$ does not contain any δ-terms, and since the interpretation function is the exact same for FOL and FOL$^\delta$ aside from the δ-terms, $I^\alpha(\sigma(A)) = 1$ in FOL$^\delta$ iff $I^\alpha(\sigma(A)) = 1$ in FOL. □

Note that this also gives us the completeness of FOL$^\delta$:

Theorem 27 (Completeness of FOL$^\delta$)**.** *FOL$^\delta$ is complete.*

Proof. We have shown the equivalence of any sentence in FOL$^\delta$ to sentences that are in FOL. FOL itself is complete, and any FOL-proof is also a proof in FOL$^\delta$, since the deduction rules of FOL are a subset of the ones of FOL$^\delta$. □

6 Conclusion

We have presented a sound and complete extension of FOL with a description operator, and have described a semantics-preserving translation into FOL. We have demonstrated its usefulness by giving a natural formalization of some parts of algebra and basic set theory. FOL$^\delta$ greatly facilitates authoring first-order theories in a way that stays close to standard math vernacular.

Future work will focus on tool support, especially for editing, theorem proving and automating the translation. Moreover, the same techniques applied here can be applied to extend FOL with a choice operator or partial functions.

References

[1] F. Horozal, F. Rabe, *Representing Model Theory in a Type-Theoretical Logical Framework*, Theoretical Computer Science, vol. 412, 2011, p. 4919-4945

[2] F. Rabe, M. Kohlhase, *A Scalable Module System*, Information and Computation vol. 230, 2013, p. 1-54

[3] P. Martin-Löf *On the Meanings of the Logical Constants and the Justifications of the Logical Laws*, Nordic Journal of Philosophical Logic, vol. 1, 1996, p. 3-10

[4] W. Howard, *The formulas-as-types notion of construction*, To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism, ed. J. Seldin and J. Hindley, Academic Press, 1980, p. 479-490

[5] R. Harper, F. Honsell, G. Plotkin, *A framework for defining logics*, Journal of the Association for Computing Machinery, vol. 40, 1993, p. 143-184.

[6] G. Sutcliffe and C. Suttner, *The TPTP Problem Library: CNF Release v1.2.1*, Journal of Automated Reasoning, vol. 21, nr. 2, 1998, p.177-203.

[7] CoFI (The Common Framework Initiative),*CASL Reference Manual*, 2004, Springer, LNCS, vol.2960.

[8] F. Rabe, *First-Order Logic with Dependent Types*, in *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, eds. N. Shankar and U. Furbach, 2006, Lecture Notes in Computer Science, vol. 4130, p. 377–391.

[9] K. Sojakova and F. Rabe, *Translating Dependently-Typed Logic to First-Order Logic* in *Recent Trends in Algebraic Development Techniques*, eds. A. Corradini and U. Montanari , 2009, vol. 5486, p. 326–341.

[10] A. Church, *A Formulation of the Simple Theory of Types*, Journal of Symbolic Logic, 1940 vol.5, p. 56–68.

[11] H. Barringer, J.H. Cheng, C.B. Jones, *A Logic Covering Undefinedness in Program Proofs*, Acta Informatica, 1984, vol. 21, p. 251–269.