Modelling the usage of partial functions and undefined terms using presupposition theory

Marcos Cramer

University of Luxembourg, Luxembourg,
marcos.cramer@uni.lu,
Web page: http://icr.uni.lu/mcramer

Abstract. We describe how the linguistic theory of presuppositions can be used to analyse and model the usage of partial functions and undefined terms in mathematical texts. We compare our account to other accounts of partial functions and undefined terms, showing how our account models the actual usage of partial functions and undefined terms more faithfully than existing accounts. The model described in this paper has been developed for the *Naproche* system, a computer system for proof-checking mathematical texts written in controlled natural language, and has largely been implemented in this system.

Keywords: partial functions, undefined terms, presuppositions, domain conditions, accommodation, Naproche

1 Introduction

Partial functions are ubiquitous in mathematical practice. For example, the division function / over the complex numbers (or any other field) is partial, since z_1/z_2 is only defined if $z_2 \neq 0$. Another prominent example is the square root function over the real numbers, which is only defined for non-negative arguments. Partial functions appear in very basic mathematics, for example the subtraction function over the natural numbers, as well as in more advanced mathematics, for example the Lebesgue integral function $\int$ over real functions, which maps Lebesgue integrable functions to their Lebesgue integrals but is undefined on other real functions. As can be seen from these examples, partial functions can be either unary or of higher arity; for simplifying the exposition, we will concentrate on unary partial functions for the rest of this introduction.

The usual notion of the *domain* of a function separates into two distinct notions in the case of partial functions: The *domain of application* of a partial function consists of the values to which it may be applied, and the *domain of definition* of a partial function consists of the values at which it is defined. For example, the square root function over the reals mentioned above has $\mathbb{R}$ as its domain of application; its domain of definition is the set $\mathbb{R}_0^+$ of non-negative reals. The domain of definition is always a subset of the domain of application. A partial function is called a *total function* if its domain of definition is identical to its domain of application.

When a mathematical expression denoting a partial function is applied to a term denoting an element outside its domain of definition, the resulting term is an *undefined term*. Thus, if $\sqrt{}$ denotes the square root function over the reals, $\sqrt{-1}$ is an undefined term. An undefined term does not refer to any mathematical object.

The common formal systems like first-order logic and simple type theory do not have any means to formalize partial functions and undefined terms in them. Standard one-sorted first-order logic allows for function symbols, but these necessarily denote total functions. Both the domain of application and the domain of definition of a function denoted by a function symbol must coincide with the *domain of discourse*, i.e. with the domain specified by the structure used for interpreting all terms and formulae in the formalism, which is also the *domain of quantification* over which the quantifiers range. In multi-sorted first-order logic as well as in simple type theory, one can have functions with different domains of application, but the domain of definition of any given function always coincides with its domain of application.

A number of formal systems have been proposed to account for the common usage of partial functions and undefined terms. Prominent examples include Michael Beeson's *Logic of Partial Terms* [1] and William Farmer's closely related *Partial First-Order Logic* (PFOL) [5], which we sketch below. A more recent approach by Freek Wiedijk and Jan Zwanenburg [13], the usage of *domain conditions* on top of standard first-order logic, comes very close to our approach; but the authors did not notice, or at least did not explicitly mention, the close relation of their approach to the linguistic theory of presuppositions, which helps clarifying some further points, as we will show in this paper.

This paper is partially based on [4], where we discussed the phenomenon of presuppositions in mathematical texts to a linguistic readership, and thereby already explained our approach to partial functions and undefined terms. In contrast to [4], this paper aims at a readership from the Formal Mathematics community, and for the first time compares our approach to approaches originating from this community.

Before explaining our own approach, we sketch William Farmer's Partial First-Order Logic in section 2.[1] In section 3 we explain the context in which we developed our approach, the proof checking algorithm of the Naproche system. In section 4 we sketch the linguistic theory of presuppositions and introduce the terminology from this theory that we will need in this paper. Section 5 contains those part of our account of how to use presupposition theory to model the usage partial functions and undefined terms that have been implemented in the Naproche system. In section 6 we sketch Freek Wiedijk's and Jan Zwanenburg's approach to use domain conditions to account for partial functions and undefined terms and compare it to our approach. In section 7, we show how our account from section 5 can be extended by making use of a detail of presupposition theory

[1] The main reason for introducing Farmer's approach before our own is that we will make use of the syntax of Partial First-Order Logic for presenting our own approach in the following sections.

that we ignored in section 5, the possibility to accommodate presuppositions. We then compare this extended account with Farmer's Partial First-Order Logic.

2 Partial First-Order Logic

In [5], William Farmer defined Partial First-Order Logic (PFOL). It allows for partial functions and undefined terms, and is based on the following three tenets:

1. Variables and constants are defined terms.
2. The application of a function to an undefined argument is undefined.
3. Formulas are always true or false. The application of a predicate is false if any of its arguments is undefined.

We now turn to the formal definition of PFOL. We will use the nowadays more usual symbols $\rightarrow$, $\leftrightarrow$ and ι where Farmer used $\supset$, $\equiv$ and I. The first two are the well known connectives from standard first-order logic; ι is used for formalizing definite description: $\iota x\ \varphi(x)$ corresponds to the natural language expression "the x such that $\varphi(x)$". Additionally to the already mentioned connectives $\rightarrow$ and $\leftrightarrow$, PFOL has the standard connectives $\neg$, $\wedge$ and $\vee$, the quantifiers $\forall$ and $\exists$ and the identity relation symbol $=$.

Further symbols are used for abbreviating PFOL formulae:

- $t\downarrow$ (read "t is defined") abbreviates $t = t$.
- $t\uparrow$ (read "t is undefined") abbreviates $\neg t = t$.
- $t_1 \simeq t_2$ (read "t_1 and t_2 are quasi-equal") abbreviates $t_1\downarrow \vee\, t_2\downarrow \rightarrow t_1 = t_2$.

Farmer defines the semantics of PFOL in a way very analogous to the standard definition of the semantics of first-order logic. Terms and formulae are interpreted in a *model*, which consists of a domain D and interpretations for the constants, function symbols and relation symbols. While the interpretations of constants and n-ary relation symbols are as in first-order logic, namely elements of D and total functions from D^n to $\{T, F\}$ respectively, the interpretations of n-ary function symbols now do no longer have to be total function from D^n to D, but may be partial functions from D^n to D (i.e. with D^n as domain of application, but any subset of D^n as domain of definition). It is straightforward to define the semantics of formulae in such a model according to the three tenets listed above.

3 Proof checking mathematical texts in the Naproche system

Before we can go on to explain our approach to partial functions and undefined terms, we first need to clarify the context in which it was developed, namely the proof checking of mathematical texts in the Naproche system.

The Naproche system (see [3] and [2]) is a computer program that can check the correctness of mathematical texts written in a *controlled natural language*, the *Naproche CNL*. A controlled natural language is a subset of a natural language defined through a formal grammar. By "checking the correctness" we mean that it tries to establish all proof steps found in the text based on the information gathered from previous parts of the text, in a similar way as a mathematician reads a (foundational) mathematical text if asked not to use his mathematical knowledge originating from other sources. For checking single proof steps, the Naproche system makes use of state-of-the-art *automated theorem provers* (ATPs). Given a set of *premises*[2] and a *conjecture*, an ATP tries to find either a proof that the premises logically imply the conjecture, or build a model for the premises and the negation of the conjecture, which shows that that they do not imply it. A conjecture together with a set of axioms handed to an ATP is called a *proof obligation*.

The Naproche system first translates an input text into a semantic representation format called *Proof Representation Structure* (PRS), an adaptation of *Discourse Representation Structures* [9], a common representation format in formal linguistics. The actual proof checking is performed on PRSs. For the sake of simplicity, we will in this paper assume that the Naproche CNL input is translated into a formal language that is syntactically identical with the language of PFOL, i.e. standard first-order syntax with an additional term construction principle for terms of the form $\iota x\ \varphi(x)$ for representing definite descriptions.

Since first-order and PFOL formulae are usually used to formally express single statements and not complete texts, we need to say some words about how complete texts are translated into this formal language. In the simplified exposition for this paper, we will leave out a number of constructs used for structuring mathematical texts, e.g. theorem-proof blocks, and concentrate on simple texts consisting of axioms, local assumptions and assertions. The concatenation of sentences is usually rendered by conjoining their respective translations with $\wedge$. A special case are axioms and local assumptions: When an axiom appears in a text, the part of the text starting at the axiom is translated by a formula of the form $\varphi \to \vartheta$, where φ is the translation of the axiom and ϑ is the translation of the text following the axiom. The translation of local assumptions, which are marked by one of the keywords "assume", "suppose" and "let" in the Naproche CNL, is similar, only that one has to take into account the *scope* of the assumption: In mathematical texts, an assumption always has a scope, which starts at the assumption and contains all assertions made under that assumptions. The end of the scope of an assumption is usually not marked in a special way in the natural language of mathematics, but in the Naproche CNL it is usually marked with a sentence starting with the keyword "thus".[3] The scope of an assumption

[2] In the ATP community, the term "axiom" is usually used for what we call "premise" here; the reason for our deviation from the standard terminology is that in the context of our work the word "axiom" means a completely different thing, namely an axiom stated inside a mathematical text that is to be checked by the Naproche system. The premises that we are considering here can originate either from axioms, from definitions or from previously proved results.

[3] The scope of an assumption also ends when the proof inside which the assumption was introduced is ended with a "Qed". But since the simplified fragment of the

is translated as $\varphi \to \vartheta$, where φ is the translation of the assumption and ϑ is the translation of the rest of the scope of the assumption. The translation of the scope of an assumption is embedded into the translation of a complete text as if the whole scope of the assumption were a single sentence.

Before explaining the treatment of partial functions and undefined terms in the proof checking, we will now first explain the basic functioning of the proof checking algorithm with total functions and without undefined terms. The proof checking algorithm keeps track of a list of first-order formulae considered to be true, called *premises*, which gets continuously updated during the checking process. Each assertion is checked by an ATP based on the currently active premises.

Below we list how the algorithm proceeds on an input formula φ depending on the form of φ. We use Γ to denote the list of premises considered true before encountering the formula in question, and Γ' to denote the list of premises considered true after encountering the formula in question. A proof obligation checking that φ follows from Γ will be denoted by $\Gamma \vdash^? \varphi$. For any given formula φ, we denote by $FI(\varphi)$ the *formula image* of φ, which is a list of first-order formulae representing the content of φ; the definitions of $FI(\varphi)$ and of the checking algorithm are mutually recursive, as specified below. (In the case of this proof checking algorithm with total functions and without undefined terms, the conjunction of the formulae in $FI(\varphi)$ is always logically equivalent to φ; this will, however, no longer be the case once we consider the extension of the proof checking algorithm to partial functions and undefined terms.)

– If φ is atomic, check $\Gamma \vdash^? \varphi$ and set Γ' to be Γ, φ.
– If φ is of the form $\varphi_1 \wedge \varphi_2$, check φ_1 with premise list Γ and φ_2 with the premise list that is active after checking φ_1; set Γ' to be the premise list that is active after checking φ_2.
– If φ is of the form $\varphi_1 \to \varphi_2$, check φ_2 with premise list $\Gamma \cup FI(\varphi_1)$ and set Γ' to be $\Gamma \cup \{\bigwedge FI(\varphi_1) \to \psi \mid \psi \in FI(\varphi_2)\}$.
– If φ is of the form $\neg\psi$, check $\Gamma \vdash^? \neg \bigwedge FI(\psi)$ and set Γ' to be $\Gamma, \neg \bigwedge FI(\psi)$.
– If φ is of the form $\varphi_1 \vee \varphi_2$ or $\varphi_1 \leftrightarrow \varphi_2$, check $\Gamma \vdash^? \bigwedge FI(\varphi_1) \vee \bigwedge FI(\varphi_2)$ or $\Gamma \vdash^? \bigwedge FI(\varphi_1) \leftrightarrow \bigwedge FI(\varphi_2)$ respectively; set Γ' to be $\Gamma, \bigwedge FI(\varphi_1) \vee \bigwedge FI(\varphi_2)$ or $\Gamma, \bigwedge FI(\varphi_1) \leftrightarrow \bigwedge FI(\varphi_2)$ respectively.
– If φ is of the form $\exists x\ \psi$ or $\forall x\ \psi$, check $\Gamma \vdash^? \exists x\ \bigwedge FI(\psi)$ or $\Gamma \vdash^? \forall x\ \bigwedge FI(\psi)$ respectively; set Γ' to be $\Gamma, \exists x\ \bigwedge FI(\psi)$ or $\Gamma, \forall x\ FI(\psi)$ respectively.

For computing $FI(\varphi)$, the algorithm proceeds analogously to the checking of φ, only that no proof obligations are sent to the ATP: The updated premise lists are still computed, and $FI(\varphi)$ is defined to be $\Gamma' - \Gamma$, where Γ is the active premise list before processing φ and Γ' is the active premise list after processing φ. This is implemented by allowing the algorithm to process a formula φ in two

Naproche CNL that we are currently considering does not contain theorem-proof blocks, we can ignore this special case as well as similar cases relating to other structural constructs of mathematical texts that we are now ignoring (e.g. case distinctions).

different modes: The Check-Mode described above for checking the content of φ, and the No-Check-Mode, which refrains from sending proof obligations to the ATP, but still expands the premise list in order to compute $FI(\varphi)$.

4 Presuppositions

Loosely speaking, a *presupposition* of some utterance is an implicit assumption that is taken for granted when making the utterance. In the linguistic literature on presupposition theory, presuppositions are generally accepted to be triggered by certain lexical items called *presupposition triggers*. Here are some common examples of presupposition triggers:

– Definite descriptions: In English, definite descriptions are marked by the definite article "the", possessive pronouns or genitives. The presupposition of a definite description of the form "the F" is that there is a unique object with property F.
– Factive verbs, e.g. "regret", "realize" and "know". For example, the presupposition of "A knows ϕ" is that ϕ holds true.
– Change of state verbs, e.g. "stop" and "begin". For example, the presupposition of "A stops doing x" is that A did x before.

In mathematical texts, most of the presupposition triggers discussed in the linguistic literature, e.g. factive verbs and change of state verbs, are not very common or even completely absent. Definite descriptions, however, do appear in mathematical texts as presupposition triggers (e.g. "the smallest natural number n such that n^2-1 is prime"). The presupposition of a definite description "the F" can be divided into two separate presuppositions: One existential presupposition, claiming that there is at least one F, and one uniqueness presupposition, claiming that there is at most one F.

Furthermore, there is a kind of presupposition trigger which does not exist outside mathematical texts: Expressions denoting partial functions. For example, the division symbol "/" triggers the presupposition that its second argument is non-zero; and the square root function over the reals triggers the presupposition that its argument is non-negative.[4]

Presupposition projection is the way in which presuppositions triggered by expressions within the scope of some operator have to be evaluated outside this scope. Consider for example the following three sentences:

$$\tfrac{1}{x+1} \in A \text{ and } x \neq 0. \tag{1}$$

[4] In Naproche, the division function / with its usual presupposition triggering features can be introduced in a proof text by a sentence of the following form: "For all real numbers x, y such that $y \neq 0$, there exists a real number $\frac{x}{y}$ such that $y \cdot \frac{x}{y} = x$." This paper is not concerned with the issue of how such partial functions are introduced, but with how they are used once they have been introduced. For more details on how partial functions are introduced, the interested reader should consult [2].

$$\frac{1}{x+1} < \frac{1}{x}. \tag{2}$$

$$\text{If } \tfrac{1}{x+1} \in A \text{ and } x \neq 0, \text{ then } \tfrac{1}{x} \in A. \tag{3}$$

We see that (1) and (3) presuppose that $x + 1 \neq 0$ and (2) presupposes that $x + 1 \neq 0$ and $x \neq 0$. So (3) inherits the presupposition of (1), but does not inherit the additional presupposition of (2). The precise way in which presuppositions project under various operators has been discussed at great length in the literature (see for example [11] and [8] for overviews of this dispute). Our formal treatment of presuppositions in mathematical texts turns out to have equivalent predictions (see [4]) about presupposition projection to Irene Heim's approach to presuppositions (see [7]).

Presupposition accommodation is what we do if we find ourselves faced with a presupposition the truth of which we cannot establish in the given context: We add the presupposition to the context, in order to be able to process the sentence that presupposes it. For example, if I say "John's wife is a philosopher" to someone who does not know that John has a wife, they will accommodate the fact that John has a wife, i.e. add this presupposition to the context in which they interpret the sentence. Note that presupposition accommodation is not always possible: If someone knows that John does not have a wife, they will not be able to accommodate the presuppositions of the example sentence. This is called *presupposition failure* and results in an inability to make sense of the sentence.

5 Proof checking with presuppositions

In this section we will describe how presuppositions can be handled in the proof checking algorithm of the Naproche system if we do not take care of the possibility of presupposition accommodation. The proof checking algorithm described in this section is called *PPC* (*Presuppositional Proof-Checking*). In section 7 we will describe how this proof checking algorithm can be adapted to allow for presupposition accommodation.

Most accounts of presupposition make reference to the *context* in which an utterance is uttered, and claim that presuppositions have to be satisfied in the context in which they are made. There are different formalizations of how a context should be conceptualized. For enabling the Naproche proof checking algorithm described in the previous section to handle presuppositions, it is an obvious approach to use the list of active premises as the context in which our presuppositions have to be satisfied.

In ordinary non-mathematical discourse, assertion usually are expected to provide new information, i.e. not to be logically implied by the available knowledge. In mathematical texts, on the other hand, assertions are expected to be logically implied by the available knowledge rather than adding something logically new to it. Because of this peculiarity of mathematical texts, both presuppositions and assertions in proof texts have to follow logically from the context.

For a sentence like "$\frac{1}{x+1}$ is negative" to be legitimately used in a mathematical text, both the fact that $x+1 \neq 0$ and the negativity of $\frac{1}{x+1}$ must be inferable from the context.

This parallel treatment of presuppositions and assertions, however, does not necessarily hold for presupposition triggers that are subordinated by a logical operation like negation or implication. For example, in the sentence "A does not contain $\frac{1}{x}$", the presupposition that $x \neq 0$ does not get negated, whereas the containment assertion does. This is explained in the following way: In order to make sense of the negated sentence, we first need to *make sense of* what is inside the scope of the negation. In order to make sense of some expression, all presuppositions of that expression have to follow from the current context. The presuppositions triggered by $\frac{1}{x}$ are inside the scope of the negation, so they have to follow from the current context. The containment assertion, however, does not have to follow from the current context, since it is not a presupposition, and since it is negated rather than being asserted affirmatively.

In our implementation, *making sense of a something* corresponds to processing it with the proof checking algorithm, whether in the Check-Mode or in the No-Check-Mode. So according to the above explanation, presuppositions, unlike assertions, also have to be checked when encountered in the No-Check-Mode.

For example, the formula representing the sentence (4) is (5).

$$A \text{ does not contain } \tfrac{1}{x}. \tag{4}$$

$$\neg\text{contains}(A, \frac{1}{x}) \tag{5}$$

When the checking algorithm encounters the negated formula, it needs to find the formula image of the formula in the scope of the negation, for which it will process this formula in No-Check-Mode. Now $\frac{1}{x}$ triggers the presupposition that $x \neq 0$, which has to be checked despite being in No-Check-Mode. So we send the proof obligation (6) to the ATP.

$$\mathit{\Gamma} \vdash^? x \neq 0 \tag{6}$$

Finally, the proof obligation that we want for the assertion of sentence (4) is (7):

$$\mathit{\Gamma}, x \neq 0 \vdash^? \neg\text{contain}(A, \frac{1}{x}) \tag{7}$$

In order to get this, we need to use the non-presuppositional formula image $\{\text{contain}(A, \frac{1}{x})\}$ of the formula in the scope of the negation: The non-presuppositional formula image is defined to be the subset of formulae of the formula image that do not originate from presuppositions. When extending the above proof checking algorithm to an algorithm capable of handling presuppositions, we have to use this non-presuppositional formula image wherever we used the formula image in the original proof checking algorithm. The presupposition premises which get pulled out of the formula image have to be added to the list of premises that were active before starting to calculate the formula image (see the treatment of the presupposition premise $x \neq 0$ in (7)).

When proof checking ι terms, a new constant symbol is used in the premises that make reference to the unique object presupposed by the ι term. Consider for example sentence (8), whose translation is (9). When proof checking (9), the presuppositions triggered by the ι term trigger the proof obligations (10) and (11), and the assertion of the sentence is checked by proof obligation (12); here c is the newly introduced constant symbol that refers to the unique object presupposed by the ι term.

$$A \text{ does not contain the empty set.} \tag{8}$$

$$\neg\mathrm{contain}(A, \iota x\ \mathrm{empty}(x) \wedge \mathrm{set}(x)) \tag{9}$$

$$\Gamma \vdash \exists x(\mathit{empty}(x) \wedge \mathit{set}(x)) \tag{10}$$

$$\Gamma \cup \{\mathit{empty}(c) \wedge \mathit{set}(c)\} \vdash \forall y(\mathit{empty}(y) \wedge \mathit{set}(y) \to y = c) \tag{11}$$

$$\Gamma \cup \{\mathit{empty}(c) \wedge \mathit{set}(c), \forall y(\mathit{empty}(y) \wedge \mathit{set}(y) \to y = c)\} \vdash \neg\mathit{contain}(A, c) \tag{12}$$

When presuppositions are triggered inside the scope of a quantifier, a somewhat more sophisticated approach is needed; see [4] or [2].

6 First-order logic with domain conditions

In [13], Wiedijk and Zwanenburg introduced an approach to partial functions and undefined terms which they termed *first-order logic with domain conditions*. Their approach turns out to be equivalent to *PPC*, only that their approach does not admit terms representing definite descriptions.[5] However, in the next section we will discuss how the possibility to accommodate presuppositions affects our approach: It will turn out that granted this possibility, our approach becomes practically much more similar to Farmer's approach than to Wiedijk's and Zwanenburg's.

In Wiedijk's and Zwanenburg's approach, one can use standard first-order syntax for formalizing talk about partial functions. In order to avoid potential problems caused by undefined terms, they define a set $\mathcal{DC}(\varphi)$ of *domain conditions* for every first-order formula φ. Domain conditions are judgements of the form $\Gamma \vdash \psi$. The idea is that if one wants to prove a statement about partial functions formalized by a first-order formula φ, one should not only prove φ but also establish the judgements in $\mathcal{DC}(\varphi)$.

There is an almost perfect correspondence between the domain conditions of a formula φ and the proof obligations triggered by presuppositions in our account. In simple examples, they coincide completely. For example, (13) has domain condition (14), and its presupposition triggers the proof obligation (15) in our account:

$$x > 0 \to \frac{1}{x} > 0 \tag{13}$$

[5] We ignore the if-then-else construct that Wiedijk and Zwanenburg added to first-order logic for their approach, as they at any rate consider this addition not essential.

$$x > 0 \vdash x \neq 0 \tag{14}$$

$$x > 0 \vdash^{?} x \neq 0 \tag{15}$$

This is no surprise if one looks at the formal definition of domain conditions: Domain conditions are defined relative to a context, which is a list of first-order formulae.[6] So Wiedijk and Zwanenburg actually define $\mathcal{DC}_{\Gamma}(\varphi)$ for a context Γ and a formula φ, and the set $\mathcal{DC}(\varphi)$ of absolute domain conditions can be identified with $\mathcal{DC}_{\emptyset}(\varphi)$. The contexts in their account are updated in a similar way as the premise lists in our account. For example, $\mathcal{DC}_{\Gamma}(\varphi \to \psi)$ is defined to be $\mathcal{DC}_{\Gamma}(\varphi) \cup \mathcal{DC}_{\Gamma,\varphi}(\psi)$. Here, the context Γ gets updated to Γ, φ for calculating the domain conditions triggered inside ψ, in a similar way as in our account the premise list Γ gets updated to Γ, φ when proof-checking the ψ inside $\varphi \to \psi$.

However, there is one difference between the ways their contexts and our premise lists get updated, which we illustrate through an example. The set of domain conditions of (16) is (17), and the set of proof obligations triggered by presuppositions in (16) is (18):[7]

$$\frac{1}{x} > 0 \to \frac{1}{x+1} > 0 \tag{16}$$

$$\{\vdash x \neq 0; \frac{1}{x} > 0 \vdash x + 1 \neq 0\} \tag{17}$$

$$\{\vdash x \neq 0; x \neq 0, \frac{1}{x} > 0 \vdash^{?} x + 1 \neq 0\} \tag{18}$$

As the proof checking algorithm processes $\frac{1}{x} > 0$, the premise list gets updated not only by $\frac{1}{x} > 0$, but also by its presupposition $x \neq 0$. Hence this presupposition additionally appears among the premises of the second proof obligation in (18), whereas it does not appear on the left hand side of the second domain condition in (17) according to the above cited definition of the domain conditions of an implication. But $x \neq 0$ must at any rate be provable because of the first domain condition and first proof obligation in (17) and (18) respectively, so that this syntactic difference is semantically irrelevant.

Since the domain conditions of a formula are thus semantically equivalent to the proof obligations triggered by presuppositions in our account, Wiedijk's and Zwanenburg's account is essentially equivalent to our account as described so far.

[6] Actually, Wiedijk and Zwanenburg define a context to be a list of variables and first-order formulae which satisfies the condition that all free variables in a formula in the context are previously listed as variables in the context. However, dropping the variables from their contexts does not alter their account in relevant way.

[7] We use semicolons to separate the elements in these sets, since commas are already used for separating the formulae in contexts or premise lists.

7 Accommodation of presuppositions[8]

Recall that accommodating a presupposition means adding it to the context of the utterance in case we cannot establish it in this context. One commonly distinguishes between *global* and *local* accommodation of presuppositions. Global accommodation is the process of altering the global context in such a way that the presupposition in question can be justified; local accommodation on the other hand involves only altering some local context, leaving the global context untouched. It is a generally accepted principle of presupposition theory that in usual discourse, global accommodation is *ceteris paribus* preferred over local accommodation.

In the introduction, we mentioned the peculiarity of mathematical texts that new assertions do not add new information (in the sense of logically not inferable information) to the context. Here "context" does not refer to our formal definition of context as a list of formulae. Instead, the *context* of a sentence in a mathematical texts should now be understood to be the set of models in which the axioms, definitions and assumptions in whose scope the sentence is made hold. This definition of *context* is analogous to the definition of *context* in various linguistic theories (e.g. Heim's theory of presupposition [7]) as a set of possible worlds that are under consideration when an utterance is made. When mathematicians state axioms, they limit the context, i.e. the set of models they consider, to the set where the axioms hold. Similarly, when they make local assumptions, they temporarily limit the context. But when making assertions, these assertions are thought be logically implied by what has been assumed and proved so far, so they do not further limit the context.

The modification of the context in the case of local assumptions is certainly a modification of the local context. For the sake of giving a unified treatment, it is useful to view the modification of the context in the case of axioms also as a modification of the local context, only that the mathemtician is planning to stay in this locally modified context for the rest of the text. With this understading of local as opposed to global contexts, one may succinctly state the pragmatic principle mentioned above in terms of contexts as follows: In a mathematical text, the global context may not be altered.

This pragmatic principle implies that global accommodation is not possible in mathematical texts, since global accommodation implies adding something new to the global context. Local accommodation, on the other hand, is allowed, and does occur in real mathematical texts:

> Suppose that f has n derivatives at x_0 and n is the smallest positive integer such that $f^{(n)}(x_0) \neq 0$.
> [12]

This is a local assumption. The projected existential presupposition of the definite description "the smallest positive integer such that $f^{(n)}(x_0) \neq 0$" is that

[8] Unlike our approach as discussed in section 5, the adaptations to our approach presented in this section have not yet been implemented in the Naproche system.

for any function f with some derivatives at some point x_0, there is a smallest positive integer n such that $f^{(n)}(x_0) \neq 0$. Now this is not valid in real analysis, and we cannot just assume that it holds using global accommodation. Instead, we make use of local accommodation, thus adding the accommodated fact that there is a smallest such integer for f to the assumptions that we make about f with this sentence.

When we accommodate a presupposition locally in this way, it no longer triggers a proof obligation. Instead, the content of the presupposition is added to the formula image of the atomic formula that triggered the presupposition.

7.1 Two ways to handle accommodation in presuppositional proof checking

We now consider two possible ways of handling accommodation of presuppositions in a proof checking algorithm:

- According to the linguistic theory of presuppositions, accommodation should only be performed if necessary. In the case of our proof checking algorithm, this means that we should always first try to establish the proof obligation triggered by the presupposition. If that fails, we accommodate the presupposition on the most local level possible (i.e. in the scope of the atomic formula inside which the presupposition was triggered). We call the proof checking algorithm that handles accommodation in this way *PPC+FlexAcc* (*Presuppositional Proof-Checking with Flexible Accommodation*).
- In order to compare presuppositional proof checking with Farmer's PFOL, we will additionally consider a proof checking algorithm called *PPC+ImmAcc* (*Presuppositional Proof-Checking with Immediate Accommodation*), which works as follows: *PPC+ImmAcc* does not produce any proof obligations from presuppositions, but always directly accommodates presuppositions on the most local level possible.

First we want to point out that *PPC+ImmAcc* has the same implications as Farmer's account:[9] In the case of an atomic formula $R(t_1, \ldots, t_n)$, the presuppositions triggered by the arguments $t_1, \ldots, t_n$ would become part of the the formula image of $R(t_1, \ldots, t_n)$, so that this formula image has precisely the semantics that Farmer gives to atomic formulae: It can only be true if all the presuppositions triggered by the arguments $t_1, \ldots, t_n$ hold, i.e. only if $t_1, \ldots, t_n$ are all defined terms.

Unlike *PPC+ImmAcc*, *PPC+FlexAcc* treats accommodation in the same way as does the linguistic theory of presuppositions. As we will illustrate through an example from a real mathematical text, we believe that the linguistically motivated proof checking algorithm *PPC+FlexAcc* gives rise to a better account of

[9] Since Farmer does not actually define a proof checking algorithm, we need to make precise what we mean by this: We mean that if one were to define a proof checking algorithm on PFOL in a canonical way, i.e. in a way completely analogous to the way we defined the non-presuppositional proof checking algorithm over standard first-order logic in section 3.

how mathematicians actually use potentially undefined terms arising from partial functions and definite descriptions than does Farmer's PFOL or *PPC+ImmAcc*.

7.2 *PPC+FlexAcc* and *PPC+ImmAcc* compared on an example

As an example to compare *PPC+FlexAcc* and *PPC+ImmAcc*, we use the proof of Theorem 2 of Edmund Landau's *Foundations of Analysis* [10]:

> **Theorem 2:** $x' \neq x$.
> **Proof:** Let $\mathfrak{M}$ be the set of all x for which this holds true.
> I) By Axiom 1 and Axiom 3, $1' \neq 1$; therefore 1 belongs to $\mathfrak{M}$.
> II) If x belongs to $\mathfrak{M}$, then $x' \neq x$, and hence by Theorem 1, $(x')' \neq x'$, so that x' belongs to $\mathfrak{M}$.
> By Axiom 5, $\mathfrak{M}$ therefore contains all the natural numbers, i.e. we have for each x that $x' \neq x$.

The first sentence of the proof is an assumption, which is marked by the keyword "Let". It introduces a new symbol $\mathfrak{M}$ to the discourse and at the same time assumes that this symbol refers to the same object as the definite description "the set of all x for which this $[x' \neq x]$ holds true".[10] The newly introduced symbol has a scope, which is the part of the text in which it may be used. The scope of a symbol introduced in an assumption generally coincides with the scope of the assumption. Since $\mathfrak{M}$ is still used in the last sentence of the proof, this sentence still belongs to the scope of the assumption. Since the proof inside which the assumption was made ends at that sentence, this must be the last sentence in the scope of the assumption.

According to the explanations we gave for translating texts with assumptions into the formal representation language on which the proof checking algorithm is defined, the proof of this theorem gets translated into a formula ϕ of the form

$$\mathfrak{M} = \iota s\ (\mathrm{set}(s) \wedge \forall x(x \in s \leftrightarrow x' \neq x)) \to \theta \wedge (\forall n\ (n \in \mathbb{N} \to n \in \mathfrak{M}) \wedge \forall x\ x' \neq x),$$

where θ is the translation of the text between the assumption and the last sentence of the proof. The ι term in this formula triggers the presupposition that there is a unique s satisfying $\mathrm{set}(s) \wedge \forall x(x \in s \leftrightarrow x' \neq x)$.

We want to compare how *PPC+FlexAcc* and *PPC+ImmAcc* check the cited theorem-proof block. So far, we have not said anything about proof-checking theorem-proof blocks. For the sake of the current exposition, we can translate the whole theorem-proof block by $\phi \wedge \forall x\ x' \neq x$ (here ϕ is the formula representing the proof as spelled out above, and $\forall x\ x' \neq x$ is the translation of the

[10] The introduction of new symbols in assumptions can be accounted for using linguistic theories of dynamic quantifiers, e.g. Discourse Representation Theory (see [9]) or Dynamic Predicate Logic (see [6]). See [2] for details about how to use Dynamic Predicate Logic to account for this phenomenon prevalent in the language of mathematics. As pointed out in footnote 11 below, we will have to make use of one detail of this account; we nevertheless refrain from explaining this account in detail, as this would go beyond the scope of this paper.

theorem statement, which is implicitly quantified universally). This means that the proof checking algorithm will first check the translation of the proof, and then check the translation of the theorem statement using the premise list that is active at the end of checking the translation of the proof. Furthermore, we will assume that the premise list that is active before checking the theorem-proof block contains premises that encode the basic set theory that is needed for understanding the proof. This basic set theory has to be strong enough for proving the presupposition triggered by the ι term.

In both *PPC+FlexAcc* and *PPC+ImmAcc*, $\mathfrak{M} = \iota s\ (\text{set}(s) \wedge \forall x (x \in s \leftrightarrow x' \neq x))$ will be processed in No-Check-Mode. In *PPC+FlexAcc*, the presupposition triggered by the ι term will be checked immediately after the ι term has been parsed. By our assumption about basic set theory being encoded in the premise list, the presupposition will be successfully checked. Let us assume that the rest of the proof is checked successfully. The premise list that is active right after checking the last sentence in the proof contains the formula $\forall x\ x' \neq x$. One might be tempted to think that this premise is therefore contained in the premise list that is active when checking the theorem assertion $\forall x\ x' \neq x$, in which case checking the theorem assertion would become trivial. However, note that after checking the (translation of the) last sentence of the proof, the scope of the assumption closes. After closing this assumption, the active premise list no longer contains the premise $\forall x\ x' \neq x$, but instead contains $\forall \mathfrak{M}\ (\mathfrak{M} = c \rightarrow \forall x\ x' \neq x)$, where c is the constant symbol newly introduced due to the ι term.[11] This premise trivially implies $\forall x\ x' \neq x$, so the theorem assertion still follows trivially from the premise list that is active after checking the proof.

When reading "for every x, $x' \neq x$" at the end of the proof, a mathematical reader would have the feeling that the proof is finished, since this is what had to be established. The reason why we intuitively feel that the proof is already finished when Landau writes "for every x, $x' \neq x$" is that we do not really feel the assumption at the beginning of the proof as an assumption, but just as an introduction of the temporary constant $\mathfrak{M}$ with a defined meaning. We can account for this intuition in the framework of the theory developed in this paper as follows:

The non-presuppositional content of the assumption, represented by the premise $\mathfrak{M} = c$, is trivial. After retracting the assumption, we have the premise $\forall \mathfrak{M}\ (\mathfrak{M} = c \rightarrow \forall x\ x' \neq x)$ in the active premise list. Since the non-presuppositional content of the assumption is trivial and leads to this premise trivially equivalent to the desired result, we do not feel the assumption to have any content at all, and hence do not feel it to be an assumption in the first place.

Let us now consider how *PPC+ImmAcc* processes the theorem-proof block under consideration. In *PPC+ImmAcc*, the presupposition ψ of the ι term will not be checked when processing the assumption, but will be added to the premise

[11] According to the rules for proof checking an implication, the premise list that is active after closing the assumption would have to contain a formula of the form $\mathfrak{M} = c \rightarrow \forall x\ x' \neq x$. The additional quantifier $\forall \mathfrak{M}$ comes from the account of introducing new symbols that was mentioned in footnote 10 above.

list together with the premise $\mathfrak{M} = c$ that expresses the non-presuppositional content of the assumption. At the end of the proof, we then have $\forall \mathfrak{M}\ (\psi \wedge \mathfrak{M} = c \rightarrow \forall x\ x' \neq x)$ instead of $\forall \mathfrak{M}\ (\mathfrak{M} = c \rightarrow \forall x\ x' \neq x)$ in the active premise list. Now using the fact that ψ follows from the basic set theory being encoded in the initial premise list, we will still be able to deduce $\forall x\ x' \neq x$ from the premise list that is active when processing the theorem assertion. But now the usage of the assumed basic set theory to establish ψ is at a point in the proof where a mathematical reader would not feel that anything needs to be proved. So on this account, we get a wrong prediction as to where in a proof certain facts should be established.

Besides being theoretically unappealing, such wrong predictions can also make the proof checking less feasible: It may happen that the position of the proof where the second account wrongly requires a presupposition to be established is not a trivial deduction step as in the above example, but requires some reasoning. In *PPC+ImmAcc*, this would mean that the presupposition has to be established together with this additional reasoning needed at that step, in a single proof obligation. This might make the proof obligation too hard to be established by the ATP used by the system. At the assumptions where presuppositions have to be established in *PPC+-FlexAcc*, on the other hand, one never needs to establish something else at the same time.

8 Conclusion

We have shown how the linguistic theory of presuppositions can be used to analyse and model the usage of partial functions and undefined terms in mathematical texts. We have considered three possible proof checking algorithms taking care of presuppositions, *PPC*, *PPC+ImmAcc* and *PPC+FlexAcc*, which differ in the way they treat the phenomenon of presupposition accommodation. Wiedijk's and Zwanenburg's account involving *domain conditions* is essentially equivalent to *PPC*, i.e. to not allowing presuppositions to be accommodated. Farmer's Partial First-Order Logic is equivalent to *PPC+ImmAcc*, i.e. to locally accommodating all presuppositions, without checking first whether they could be discharged. *PPC+FlexAcc* treats accommodation in the same way as linguistic presupposition theory, i.e. accommodates presuppositions only if they cannot be discharged. We have argued that *PPC+FlexAcc* provides an account of the usage of partial functions and undefined terms that models the intuitions and actual usage by mathematicians more faithfully than Farmer's or Wiedijk's and Zwanenburg's account.

References

1. Beeson, M.: Foundations of Constructive Mathematics. Springer, Berlin (1985)
2. Cramer, M.: Proof-checking mathematical texts in controlled natural language. PhD thesis, University of Bonn (2013)

3. Cramer, M., Fisseni, B., Koepke, P., Kühlwein, D., Schröder, B., Veldman, J.: The Naproche Project – Controlled Natural Language Proof Checking of Mathematical Texts. In: Fuchs, N. (ed.) Controlled Natural Language, LNCS, vol. 5972, pp. 170-186. Springer, Berlin (2010).
4. Cramer, M., Kühlwein, D., Schröder, B.: Presupposition Projection and Accommodation in Mathematical Texts. In: Pinkal, M., Rehbein, I., Schulte im Walde, S., Storrer, A. (eds.) Semantic Approaches in Natural Language Processing: Proceedings of the Conference on Natural Language Processing 2010 (KONVENS),pp. 29-36. Universaar, Saarbrcken (2010).
5. Farmer, W.: Reasoning about partial functions with the aid of a computer. Erkenntnis 43, 279-294 (1995)
6. Groenendijk J., Stokhof, M.: Dynamic Predicate Logic. Linguistics and Philosophy 14(1), 39-100 (1991)
7. Heim, I.: On the projection problem for presuppositions. In: Barlow, M., Flickinger D., Westcoat, M. (eds.) Proceedings of the Second West Coast Conference on Formal Linguistics, pp. 114-125 (1983)
8. Kadmon, N.: Formal Pragmatics. Wiley-Blackwell, Oxford, UK (2001)
9. Kamp, H., Reyle, U.: From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Languge. Kluwer Academic Publishers, Dordrecht, Netherlands (1993)
10. Landau, E.: Foundations of Analysis. Trans. F. Steinhardt. Chealsea Publishing Company, Bronx, NY, USA (1930, trans. 1951)
11. Levinson, S. C.: Pragmatics. Cambridge University Press, Cambridge, UK (1983)
12. Trench, W. F.: Introduction to Real Analysis. Pearson Education, Upper Saddle River, NJ, USA (2003)
13. Wiedijk, F., Zwanenburg, J.: First order logic with domain conditions. In: Basin D., Wolff, B. (eds.) Theorem Proving in Higher Order Logics, TPHOLs 2003. LNCS, vol. 2758, pp. 221-237. Springer, Berlin (2003)