A Prudent Logic of Partial Functions*

Đorđe Marković[1*], Robbe Van den Eede[1,2] and Marc Denecker[1,3*]

[1]Department of Computer Science, KU Leuven, Celestijnenlaan 200 A, Leuven, 3001, Belgium.

[2]Artificial Intelligence Laboratory, Vrije Universiteit Brussel, Brussels, Belgium.

[3]KU Leuven Institute for Artificial Intelligence, Leuven.AI, Leuven, Belgium.

*Corresponding author(s). E-mail(s): dorde.markovic@kuleuven.be; marc.denecker@kuleuven.be;
Contributing authors: robbe.vandeneede@kuleuven.be;

Abstract

Partial functions are ubiquitous in Knowledge Representation applications, ranging from practical, e.g., business applications, to more abstract, e.g., mathematical and programming applications. Expressing propositions about partial functions may lead to non-denoting terms. These result in undefinedness errors and ambiguity, causing subtle modeling and reasoning problems.

In our approach, formulae are well-defined (*true* or *false*) and non-ambiguous in all structures. We develop a base extension of three-valued predicate logic, in which partial function terms are *guarded* by domain expressions, ensuring the well-definedness property despite the three-valued nature of the underlying logic. This property allows us to define the satisfaction relation of the new logic in terms of the standard two-valued logic of total functions. To tackle the verbosity of this core language, we propose different ways to increase convenience by means of disambiguating annotations and non-commutative connectives. As a practically relevant result, we prove that many different *unnesting* methods, which eliminate (partial) functions by replacing them with their graph predicates, are equivalence-preserving in the proposed language.

*This version of the article has been accepted for publication, after peer review, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record will be available online once the article is published. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms.

Furthermore, we present an extension of the logic with *definitions* of partial functions and give their semantics in terms of the existing semantics for inductive definitions of sets. Finally, we explore the connection to functional programming.

Keywords: Knowledge Representation, Partial functions, Non-denoting terms, Guards, Nested function terms, Recursive definitions

1 Introduction

Partial functions and non-denoting terms have been studied in many different domains: in linguistics, from the perspective of presupposition, see Karttunen [1]; in philosophy, where the topic culminated in big discussions between Russell [2], Frege [3], and Strawson [4]; in mathematics and logic Suppes [5], Kleene [6], Schock[7], Fitting [8]; in proof theory Gavilanes-Franco and Lucio-Carrasco [9] and de Nivelle [10]; and in computation McCarthy [11]. Partial functions enjoy many different treatments in programming languages. They have been studied in the context of formal specification languages Berezin et al.[12], Hoang [13], in declarative constraint languages Cristiá et al.[14], Frisch and Stuckey[15], and knowledge representation languages De Cat et al. [16], Cabalar [17], Balduccini [18, 19]. Good overviews of different approaches can be found in [20] by Farmer, and [5] by Suppes. In this paper, we study partial functions from the perspective of modeling and Knowledge Representation (KR). This paper extends our previous work [21] by providing a better overview of the field, further substantial development of the guards (main concept of the paper), and introducing support for recursive definitions.

The problem with non-denoting terms arises when applying a function concept to some entity for which the function is undefined. This problem can occur in natural language or mathematical text, in formal logic theories, or programs of declarative or imperative programming languages. A well-known example of a sentence with a non-denoting term, made famous by Bertrand Russell, is: “The king of France is bald” (it is non-denoting as France has no king.) In mathematical texts, the problem arises, e.g., in propositions applying division to zero or the square root to a negative number when working over the real numbers (i.e., $\frac{x}{y}$, $\sqrt{x}$). In logic, introducing partial functions breaks the two-valuedness of a logic, causing invalidity of the law of the excluded middle (a.k.a. the principle of the excluded third, or in Latin “principium tertii exclusi”, or “tertium non datur”). In modeling applications, partial functions are ubiquitous. They mainly lead to problems when the universe is heterogeneous, with many different types of objects, or when the domain of a partial function concept is unknown. An example of the latter would be an application of a function $age/1$ to a constant $myPartner/0$, while we do not know whether $myPartner$ actually exists.[1]

Many logics only support *total* functions. For example, in first-order predicate logic (FO), the value of a function symbol in a structure is by definition a total

[1]We use camel case notation (i.e., spaceless phrases with capitalized words), with the first letter capitalized for predicate symbols (e.g., $SomePredicate/n$) and lowercase for function symbols (e.g., $someFunction/n$), where n indicates the symbol's arity.

function. Nevertheless, in many applications, total functions are used to represent partial function concepts [20]. This formally circumvents the phenomenon of non-denoting terms, but doing so leads to a semantic mismatch, as illustrated by the following graph coloring example.

$$\forall x : \mathit{Vertex}(x) \Rightarrow \mathit{Color}(\mathit{colorOf}(x))$$
$$\forall x : \forall y : \mathit{Graph}(x, y) \Rightarrow \neg(\mathit{colorOf}(x) = \mathit{colorOf}(y))$$

The formulae express that the vertices are colored by a color, and that two adjacent nodes in the graph are colored differently. Conceptually, a coloring function is defined only on vertices, but in first-order logic, $\mathit{colorOf}/1$ is interpreted as a total function, and hence defined on colors as well. As a consequence, there is a *semantic mismatch* between the states of affairs that were intended by the modeler and the formal models of the logical theory. This leads to subtle representation and reasoning problems. For example, applying *model generation inference* on (an extension of) this theory will return many redundant models (i.e., structures that assign the same colors to vertices but assign different colors to colors). For the same reason, *model counting inference* (in which the total number of models of a logical theory is counted) will return wrong answers. Additionally, modeling partial functions by means of total functions may lead to subtle modeling errors. For instance, to express that all colors are to be used for coloring, a modeler may write a sentence $\forall x : \mathit{Color}(x) \Rightarrow \exists y : \mathit{colorOf}(y) = x$. However, this would be incorrect, as it is necessary to condition y to be a vertex.

Experienced knowledge engineers develop practices to circumvent these issues, such as:

1. Using the predicate *Vertex* to only apply the *colorOf* function to vertices, on which *colorOf* is defined. This approach requires additional efforts during modeling, as demonstrated in the example above.
2. Representing the function concept $\mathit{colorOf}/1$ as a *graph predicate* $\mathit{HasColor}/2$, and adding additional formulae to strengthen theories or axioms (e.g., to formalize uniqueness). This approach leads to more verbose theories. For instance, $\mathit{colorOf}(a) \neq \mathit{colorOf}(b)$ would become $\neg\exists c : \mathit{HasColor}(a, c) \wedge \mathit{HasColor}(b, c)$.
3. Introducing a new "null" object to the domain of colors to *simulate the absence* of color. Such a pollution of the domain is dangerous, as this artificial null object is included in the range of quantifiers.
4. Switching to a *sorted logic*, and turning the *Vertex* predicate into a type. This resolves the semantic mismatch because the function *colorOf* becomes a total on the *Vertex* type. However, this easily leads to an unwieldy collection of types. If we were to only color the vertices with an outgoing edge, for instance, this would require a type for vertices with an outgoing edge.

With such techniques and efforts, the problems caused by the semantic mismatch can be resolved, but the semantic mismatch does not necessarily disappear. Furthermore, handling partial functions with the aforementioned techniques introduces new challenges and requires additional efforts. This forms an argument for providing a systematic treatment of partial functions in modeling languages. A systematic treatment of partial functions is especially important for Knowledge Representation languages, where addressing semantics mismatch is crucial, and partial functions are present in virtually any domain.

In this paper, we approach statements involving partial functions as inherently ambiguous. This ambiguity arises from the difficulty of assigning a truth value to propositions about non-existent objects. We demonstrate this with Russell's example: "The king of France is bald". Intuitively, in the present state of affairs, this sentence seems to be false (given that there is no third option besides the sentence being true or false). In his work [2], Russell claims that the statement is false and justifies it by interpreting it as: "*There exists an x which is the king of France and x is bald*". However, consider the statement "The wife of the king of France is the queen of France". Intuitively, this statement is a tautology, even though France has neither a king nor a queen at the moment. It thus seems that this sentence is interpreted[2] as: "*For all x and y, if x is the king of France, then if y is the wife of x then y is the queen of France*". Arguably, the same statement "The king of France is bald" could be interpreted as "*For all x, if x is the king of France then x is bald*" in a certain context. Imagine, for instance that the statement is found in the medieval French parliamentary charter in which it is decreed that the king of France is to be bald.[3] Hence, an atomic statement $P(t)$ where t is potentially non-denoting, could be interpreted[4] as "*There exists an x such that x is t and $P(x)$*" or as "*For each x, if x is t, then $P(x)$*". If the term t denotes, these two interpretations coincide, and hence the meaning of the statement is clear. However, if t does not denote, the two interpretations are not equivalent, and hence the original statement $P(t)$ can be perceived as ambiguous. One goal of this paper is to design a logical framework for effectively representing disambiguated forms of such statements. We refer to this as the *prudence* approach, as the syntax of the languages necessitates explicit elimination of ambiguities. Given that any non-denoting term introduces ambiguity, achieving prudence quickly becomes unfeasible. Therefore, another key contribution of this paper is the development of a language specifically designed for a compact and precise elimination of ambiguities.

We already pointed out that the problem of non-denoting terms in modeling languages emerges from propositions including partial functions. These partial functions could be part of the language (e.g., division) or declared by the user. In the standard logical setup, one can constrain the values of a function, but there are no means to express how such mathematical objects are constructed. However, it is often the case that user-introduced functions can be defined in terms of other concepts or in terms of themselves (recursive functions, e.g., Fibonacci sequence). This idea is parallel to

[2]A similar interpretation is achieved if "the king of France" is treated not as a denoting expression, but as a concept, which is in line with Russell's theory of descriptions.

[3]It is possible to interpret this statement as obligatory (deontic), which is not our goal. Rather, we merely want to point out that interpretations (universal or existential) might differ depending on the context.

[4]We observed in different examples that definitional and general law statements are usually interpreted with universal quantifiers and implications, while empirical or assertive statements are often interpreted using existential quantifiers and conjunctions. This distinction should be understood as a heuristic rather than a strict rule, as there are exceptions depending on context and interpretation, as demonstrated with "the king of France" example.

the one of extending logic with inductive definitions of sets [16, 22]. Hence, another goal of this paper is to introduce logic supporting recursive definitions of functions.

The main contribution of this paper is the development of a logic FO(PF) for partial functions, including recursive definitions of functions. The leading principle is that the logical treatment of partial functions has to result in formulae that are well-defined (*true* or *false*) and non-ambiguous in all structures. The logic should cope with partial functions with an unknown domain. Moreover, the well-definedness of logical expressions should be a decidable property. A first basic approach extends 3-valued predicate logic with guards, in which partial function terms are guarded by domain expressions ensuring the well-definedness property despite the 3-valued nature of the underlying logic. Next, to tackle the verbosity of this core language, we propose different ways to increase convenience by using disambiguating annotations and non-commutative connectives. Additionally, the language is extended with recursive definitions of functions. We show that the proposed logic is reducible to the two-valued logic of total functions. Furthermore, we prove that many different unnesting methods turning partial functions into graph predicates, which are not equivalence preserving in general, are equivalence preserving in the proposed language, showing that ambiguity is avoided.

The paper is structured as follows: First, we provide a detailed overview of various fields related to partial functions. Next, the syntax and semantics of FO(PF) are defined. Subsequently, we propose the language of guards, show its semantic reduction to the total function logic, and present convenient guard constructs. Next, we demonstrate the properties of guarded logic in terms of the elimination of function terms. Finally, we dedicate special attention to (recursive) definitions of partial functions and prove a correspondence result between our definition logic and a formalism used in functional programming to define partial functions by means of lambda expressions. We close the paper with the conclusion.

2 Related work

Non-denoting terms are studied in many fields, including linguistics, philosophy, mathematics, and computer science. This section provides an overview of these fields.

Linguistics. From the perspective of *linguistics*, non-denoting expressions are tightly related to presuppositions [1]. For example, the sentence "My partner is a great volleyball player" is only sensible if the presupposition is made that "My partner" exists and is unique. Interestingly, presuppositions are sometimes preserved under negation, e.g., "My partner is not a great volleyball player" still presupposes that "My partner" exists. Here, we highlight the two main points from [1] relevant to our work: 1) Presuppositions in a natural language sentence can differ depending on the context. 2) A conditional connective may be useful to eliminate presuppositions, e.g., the sentence "*If* my partner exists, *then* my partner is a great volleyball player" does not make a presupposition.

Philosophy. The issues caused by denotation (e.g., failure of the law of excluded middle) lead to many different *philosophical* views, perhaps most famously those of Frege [3] (and Strawson [4]) and Russell [2]. Russell argued that a statement such as "My partner is a great volleyball player" should be interpreted as "There exists exactly one person that is my partner, and this person is a great volleyball player", thus making the presupposition explicit. Furthermore, he argued that the negation of this sentence should be interpreted as either "*It is false that* there exists exactly ..." or "There exists exactly ..., and this person is *not* a great volleyball player". In case "My partner" does not exist or is not unique, the former statement is *true* and the latter *false*. On the other hand, according to Strawson [4], the truth of statements of the form "My partner ..." is simply not defined in a state of affairs where the partner does not exist. Notice that following this approach in a formal logic would require a three-valued truth assignment.

Mathematics. In *mathematical texts*, the problem of non-denoting terms emerges in applications of partial functions such as subtraction in the natural numbers and division in any set of numbers (the divisor must be different than zero).[5] Mathematical texts do not (or much less) tolerate ambiguities and implicit presuppositions, leading to the establishment of strong principles for treating partial functions. Several works [5, 23] have studied how partial functions are dealt with in mathematics. In [5, Section 8.5] Suppes outlines the problem of defining the division operator in terms of multiplication (i.e., $x/y = z$ *if and only if* $x = y \times z$). The problems appear for $y = 0$. First, for $x = 0$, the uniqueness condition does not hold since $0 = 0 \times z$ holds for any z. Second, for $x \neq 0$, the existence condition does not hold since there is no z such $0 \times z \neq 0$. Furthermore, in [5, Section 8.6] Suppes discusses *Conditional Definitions* which are relevant for our Section 7. Our goal is to develop a formal logic treatment of partial functions that achieves the same rigor as (informal) mathematical text.

Three-valued logic. An overview of the history of three-valued logics is available in [24] by Cobreros et al. Probably the most famous formalism is *Kleene's three-valued logic* [6, page 332], which was inspired by problems of mathematical foundations and recursion theory. Kleene introduced the principle of *regularity*, meaning that if the truth value of any constituent of a statement changes from undefined to true or false, then the truth value of the statement should not change from true to false or from false to true (but it may change from undefined to true or false). Among the different possibilities for logical connectives that satisfy this property, Kleene noticed that there is a weakest and a strongest one. In the weak version, a conjunction is undefined if any of its conjuncts is undefined, while in the strong version, a conjunction is false if one conjunct is false and the other is undefined. In this paper, Kleene's strong three-valued logic is used for the semantics of the proposed language (Definition 5). In [11], McCarthy introduced a *left-sequential three-valued propositional logic* to create mathematical bases for computation. The special aspect of this logic is that its semantics can be associated with the order of evaluation from left to right. For example, the statement $\mathit{true} \vee (1/0 = 10)$ is true, while $(1/0 = 10) \vee \mathit{true}$ is undefined. The logical connectives in such a language are called asymmetric. In our paper, connectives similar to those of Kleene's weak logic and McCarthy's left-sequential logic are used to provide convenient expressions of implicatures regarding partial functions (Section 5.1).

[5]For an extensive list of interesting mathematical riddles demonstrating problems with division by zero, we refer to the book [23, Chapter 5] by Northrop and Silver.

A *free logic* (for details, check [25, Chapter "Free logics"]) is a logic deprived of any existential assumptions. This means that terms do not have to denote any object and models with empty domains are allowed (a.k.a. *inclusive logic*). *Free logics* are mainly studied in a first-order setting, and depending on the treatment of the non-denoting terms, they are classified as: *negative*, where all atoms containing empty terms are false, *positive*, where some atoms containing empty terms are true, and *neutral*, where all atoms containing empty terms are valueless. In this paper, the formal semantics of our proposed language corresponds to the neutral semantics (Definition 5). However, we shall show that the guarding conditions of our logic ensure that each of the three semantics is equivalent (following from Theorem 1 and Theorem 2). An important language construct that is customary for *free logics* is the "existence" predicate (common syntax: $E!\ t$ or simply $E\ t$). The existence predicate applied to a term is true if that term denotes an element of the domain, and false otherwise. For *intuitionistic logic*[6], Scott defined $E\ t$ in his *Logic of Existence* [29] as $(t = t)$. In this paper, we use a similar "definedness" predicate δ, for example, $\delta_/(1, 0)$ evaluates to false since $1/0$ does not denote. The main difference is that we require terms applied to δ to be defined (e.g.., $\delta_/(1, (1/0))$ is undefined since $1/0$ is undefined).

In [20], Farmer extends Church's *simple theory of types* [30] to include partial functions by allowing functions to evaluate to undefined, while atomic formulae applied to undefined terms evaluate to false. In *dependent type theory*, the type of a function can contain additional information, for example, division can be declared as a function that takes a number as its first argument, a non-zero number as its second argument, and returns a number (for practical details consult [31, Section 2.7 and Section 7] by Bove and Dybjer). With such typing, one must prove that the divisor is not zero for every use of division, while in our approach, this is guaranteed by the syntactic constraints of our language.

Proof theory. In [10], de Nivelle introduces extensions of *Classical Logic* with partial functions where the semantics is designed to ensure that formulae containing a non-denoting term do not have a truth value. This approach interprets ill-defined formulae as *errors*. A sound and complete sequent calculus is defined for this language. This formalism is specific for lazy operators (*lazy implication* $[A]B$ and *lazy conjunction* $\langle A \rangle B$), and for bringing type declarations of symbols to the theory level. Laziness means that the second proposition B will be ignored when the first one, A, is *false*. In this way, one can write statements in which irrelevant parts are undefined. For example, the type and the preconditions of the subtraction function, represented with the "$-$" symbol, is declared using *lazy implication*: $\forall x, y : [Nat(x) \wedge Nat(y)]\,(x \geq y) \Rightarrow Nat(x - y)$. In natural language, this expression says that subtraction is defined for natural numbers only when the first argument is greater than or equal to the second.

Gavilanes-Franco and Lucio-Carrasco [9] investigate first-order logic with partial functions. The language proposed in that paper is close to standard first-order logic, with the addition of conditional terms and a definedness meta-predicate (Φ). Applying functions and predicates to undefined terms results in undefinedness, except for the definedness predicate Φ. Furthermore, a weak and a strong semantics are defined,

[6]Early studies of intuitionistic logic resulted in three-valued logics such as: Heyting's *intuitionistic propositional calculus* [26] (a.k.a. *logic of here and there* Pearce [27]), and Gödel's logic G_3 [28].

along with four different logical consequences. Finally, the paper presents a sound and complete sequent calculus for this language.

Related formal languages and systems. The idea of eliminating (partial) function terms by introducing new existentially quantified variables and replacing the function with its graph (as described in [5, Section 8.7]) is implemented in the *Knowledge Base System IDP* [16, Section 1.2.2] by De Cat et al. and in *MiniZinc* [15] by Frisch and Stuckey. According to this method, the statement $div(10, 0) = 1$ would be interpreted as $\exists y : \gamma_{div}(10, 0, y) \land (y = 1)$. Here, γ_{div} denotes the graph of the division function, which is false for any triple (x, y, z) with $y = 0$. Consequently, the translated formula is false. In case users had another disambiguation in mind, they have to explicitly specify it by eliminating the function term, e.g., $\forall y : \gamma_{div}(10, 0, y) \Rightarrow (y = 1)$.

In the *Answer set programming* (ASP) community, substantial efforts have been made to integrate partial functions in ASP [17–19]. These papers aim at the proper treatment of partial functions and at constructing practical systems for ASP. In the paper by Cabalar, any atom that contains an undefined term evaluates to *false*. Furthermore, similar to Scott's existence operator, Et is introduced as a derived operator standing for $t = t$, which is false precisely when t is undefined. Given that a function applied to an undefined term is itself undefined, it follows that $Ef(t)$ implies Et. Similarly, $t_1 \# t_2$ stands for $Et_1 \land Et_2 \land \neg(t_1 = t_2)$. The most interesting operator (for our work) introduced in [17] is $[P(t_1, \ldots, t_n)]$, which stands for $Et_1 \land \cdots \land Et_n \Rightarrow P(t_1, \ldots, t_n)$. In this paper, we will introduce a similar operator in Section 5.3.

The approach to partial functions by Berezin et al. in the theorem prover *Cooperating Validity Checker* (CVC) Lite [12] is that a theory must be provably well-defined (referred to as the *principle of least surprise*). A similar approach appears in the Event-B Modeling Method [13]. In these systems, given a formula ψ, the Type Correctness Condition (TCC) ψ_{TCC} formula is generated; if ψ_{TCC} is valid, the formula ψ is well-defined. Importantly, the formula ψ_{TCC} is well-defined by construction.

3 FO(PF) Syntax

This section describes the syntax of our extension of first-order logic with partial functions. We use various meta-variables: t for terms, ϕ, ψ, χ for formulae, and α for expressions.

Definition 1 A *vocabulary* Σ is a set of predicate and functions symbols σ, each having an arity $n \geq 0$ (denoted as σ/n). A function symbol $f/0$ is called an *constant symbol*, a predicate symbol $p/0$ is called a *propositional symbol*. In addition, for each function symbol f/n, Σ contains a *domain predicate* symbol δ_f/n and a *graph predicate* symbol $\gamma_f/n + 1$.

The equality symbol $=$ is a special binary predicate symbol with a fixed interpretation (as specified in Definition 4), for which we use infix notation. Intuitively, each function symbol f has an associated *domain predicate* δ_f, expressing the *domain* of the function, i.e., where f is defined, as well as a *graph predicate* γ_f of f, which represents the set of pairs $(\bar{x}, y)$ such that the tuple $\bar{x}$ is mapped to an element y by

f. We assume that the logic possesses a countably infinite set X of variable symbols (different from predicate and function symbols).

Definition 2 Given a vocabulary Σ, an FO(PF) *term* over Σ is defined inductively as follows:

- If $x \in X$, then x is a term. We refer to such terms as *variable terms*.
- If $f/n \in \Sigma$ is a function symbol and $t_1 \ldots, t_n$ are terms over Σ, then $f(t_1, \ldots, t_n)$ is also a term. We refer to such terms as *non-variable terms*. For $n = 0$, we also call $f()$ a *constant*. By a slight abuse of notation, we will write f instead of $f()$.

Definition 2.1 Similarly, an FO(PF) *formula* over Σ is defined inductively as follows:

- The propositional symbols **t** and **f**, denoting true and false respectively, are formulae.
- If $p/n \in \Sigma$ is a predicate symbol and $t_1 \ldots, t_n$ are terms over Σ, then $p(t_1, \ldots, t_n)$ is a formula. For $n = 0$, we also call $p()$ a *proposition*. By a slight abuse of notation, we will write p instead of $p()$.
- If ϕ, ψ, χ are formulae over Σ, then so are:
$$\neg\phi \qquad \phi \vee \psi \qquad \textbf{if } \phi \textbf{ then } \psi \textbf{ else } \chi \textbf{ fi} \qquad \exists x : \phi$$

An occurrence of a variable x in a formula ϕ is said to be *bounded* in ϕ if it is in ψ for a subformula $(\exists x : \psi)$ of ϕ. Otherwise, the occurrence of x is said to be *free* in ϕ. To avoid any technical complications with the scopes of variables, we assume that no variable has both a free and a bounded occurrence in a formula, and that each quantifier introduces a different variable.

Definition 3 An FO(PF) *expression* is an FO(PF) term or a formula. An FO(PF) *sentence* is an FO(PF) formula in which every occurrence of a variable is bounded. An FO(PF) *theory* is a set of FO(PF) sentences.[7]

The FO(PF) language, as defined in Definition 2, is composed of a limited set of language connectives. Other familiar connectives, $\wedge$, $\Rightarrow$, $\forall$, and $\Leftrightarrow$ can be defined in terms of the basic connectives in the standard way, e.g., $\phi \Rightarrow \psi$ stands for $\neg\phi \vee \psi$.

A material implication $\phi \Rightarrow \psi$ is similar in meaning to the conditional **if** ϕ **then** ψ **else t fi**, but not equivalent. The difference appears, for example, in a 3-valued context in which ϕ is undefined and ψ is true. There the material implication evaluates to true and the *if-then-else* statement to undefined. The latter statement is equivalent to what is known as the asymmetric or sequential version of the material implication [8, 11]. The *if-then-else* conditional will be the building block for the *guarded expressions* that we will define in Section 5.

4 FO(PF) semantics

In this section, we define the semantics of FO(PF) by defining the value of expressions in a structure. Here, a structure is a mathematical object abstractly representing a

[7]By *theory* we do not mean the deductively closed set of sentences, but an arbitrary set of sentences.

state of affairs (i.e., value assignment to vocabulary symbols). Based on the truth conditions of the language connectives, the three-valued truth assignment to complex expressions is recursively defined.

Following is the formal definition of a FO(PF) structure. Below, D^n denotes the n-fold Cartesian product of the set D.

Definition 4 Given a vocabulary Σ, a *partial function structure* $\mathfrak{A}$ over Σ is a two-valued structure consisting of:

- A non-empty set $\mathcal{U}^{\mathfrak{A}}$, called the universe of $\mathfrak{A}$. If $\mathfrak{A}$ is clear from the context, it will be denoted compactly as $\mathcal{U}$.
- An assignment of interpretations $\sigma^{\mathfrak{A}}$ to non-logical symbols $\sigma \in \Sigma$:
  1. Per predicate symbol $p/n \in \Sigma$, a relation $p^{\mathfrak{A}} \subseteq \mathcal{U}^n$.
  2. Per function symbol $f/n \in \Sigma$, a set $f^{\mathfrak{A}} \subseteq \mathcal{U}^n \times \mathcal{U}$ such that for all tuples $(\bar{d}, e_1), (\bar{d}, e_2) \in f^{\mathfrak{A}}$, it holds that $e_1 = e_2$. The corresponding relation[8] is assigned to the graph predicate symbol ${\gamma_f}^{\mathfrak{A}} = \{(d_1, \dots, d_n, e) \mid ((d_1, \dots, d_n), e) \in f^{\mathfrak{A}}\}$. If $((d_1, \dots, d_n), e) \in f^{\mathfrak{A}}$, we write $f(d_1, \dots, d_n) = e$.
  3. Per domain predicate symbol $\delta_f/n \in \Sigma$, ${\delta_f}^{\mathfrak{A}}$ is the domain of $f^{\mathfrak{A}}$, i.e., $\{\bar{d} \mid \exists e \in \mathcal{U} : (\bar{d}, e) \in f^{\mathfrak{A}}\}$.
- The interpretation $=^{\mathfrak{A}}$ is the identity relation $\{(d, d) \mid d \in \mathcal{U}\}$ on $\mathcal{U}$.[9]

To be able to evaluate formulae with free variables, a structure $\mathfrak{A}$ needs to be extended with a variable assignment denoted here as $[x_1 : d_1, \dots, x_n : d_n]$. Thus, $\mathfrak{A}[x_1 : d_1, \dots, x_n : d_n]$ denotes the structure obtained from $\mathfrak{A}$ by interpreting the variable x_i by a domain element $d_i \in \mathcal{U}$ for all i. To denote the value d_i of a variable x_i in $\mathfrak{A}$, we use the $x_i^{\mathfrak{A}}$ notation.

We now define the *evaluation function* of FO(PF) as an extension of Kleene's strong truth assignment. It maps expressions to their value. A term is mapped to either an object from the domain of discourse or to the special value $\bot_{term}$ representing an undefined object. A formula is mapped to $\mathbf{t}$, $\mathbf{f}$, or the special value $\bot_{formula}$ representing an undefined truth value. Hence, while the structure is two-valued, the evaluation function is 3-valued.[10] When clear from the context, we omit subscripts and simply write $\bot$.

Definition 5 Let α be an FO(PF) expression over vocabulary Σ, and $\mathfrak{A}$ a partial function structure (over vocabulary Σ) extended with a variable assignment over the free variables of α. The value of α in $\mathfrak{A}$, denoted as $[\![\alpha]\!]^{\mathfrak{A}}$, is defined by induction on the structure of α:

$$[\![x]\!]^{\mathfrak{A}} = x^{\mathfrak{A}}.$$

[8]The function and its graph symbol do not share the same interpretation within the structure (although they contain the same information), as this distinction simplifies certain definitions and theorems later in the paper.

[9]Note that every structure with the same domain interprets "=" as the same relation. In certain contexts, it is natural to include additional symbols with a fixed interpretation, e.g., in arithmetic, one could add function symbols for the operators $+, -, \cdot, /$ with their standard interpretations, together with the corresponding domain and graph predicates.

[10]An alternative would be to incorporate the undefined values in our notion of structure. However, we choose to stay close to the standard set-theoretic implementation of concepts.

$$[\![f(t_1,\dots,t_n)]\!]^{\mathfrak{A}} = \begin{cases} f^{\mathfrak{A}}([\![t_1]\!]^{\mathfrak{A}},\dots,[\![t_n]\!]^{\mathfrak{A}}), \text{if } [\![t_i]\!]^{\mathfrak{A}} \neq \bot \text{ for } i \in \{1,\dots,n\} \\ \qquad \text{and } ([\![t_1]\!]^{\mathfrak{A}},\dots,[\![t_n]\!]^{\mathfrak{A}}) \in \delta_f^{\mathfrak{A}}; \\ \bot, \quad \text{otherwise;} \end{cases}$$

$$[\![p(t_1,\dots,t_n)]\!]^{\mathfrak{A}} = \begin{cases} \mathbf{t}, & \text{if } [\![t_i]\!]^{\mathfrak{A}} \neq \bot \text{ for } i \in \{1,\dots,n\} \text{ and } ([\![t_1]\!]^{\mathfrak{A}},\dots,[\![t_n]\!]^{\mathfrak{A}}) \in p^{\mathfrak{A}}; \\ \mathbf{f}, & \text{if } [\![t_i]\!]^{\mathfrak{A}} \neq \bot \text{ for } i \in \{1,\dots,n\} \text{ and } ([\![t_1]\!]^{\mathfrak{A}},\dots,[\![t_n]\!]^{\mathfrak{A}}) \notin p^{\mathfrak{A}}; \\ \bot, & \text{otherwise;} \end{cases}$$

$$[\![\neg\phi]\!]^{\mathfrak{A}} = \begin{cases} \mathbf{t}, & \text{if } [\![\phi]\!]^{\mathfrak{A}} = \mathbf{f}; \\ \mathbf{f}, & \text{if } [\![\phi]\!]^{\mathfrak{A}} = \mathbf{t}; \\ \bot, & \text{if } [\![\phi]\!]^{\mathfrak{A}} = \bot. \end{cases}$$

$$[\![\exists x : \phi]\!]^{\mathfrak{A}} = \begin{cases} \mathbf{t}, & \text{if for some } d \in \mathcal{U}^{\mathfrak{A}}, [\![\phi]\!]^{\mathfrak{A}[x:d]} = \mathbf{t}; \\ \mathbf{f}, & \text{if for all } d \in \mathcal{U}^{\mathfrak{A}}, [\![\phi]\!]^{\mathfrak{A}[x:d]} = \mathbf{f}; \\ \bot, & \text{otherwise;} \end{cases}$$

$$[\![\phi \vee \psi]\!]^{\mathfrak{A}} = \begin{cases} \mathbf{t}, & \text{if } [\![\phi]\!]^{\mathfrak{A}} = \mathbf{t} \text{ or } [\![\psi]\!]^{\mathfrak{A}} = \mathbf{t}; \\ \mathbf{f}, & \text{if } [\![\phi]\!]^{\mathfrak{A}} = [\![\psi]\!]^{\mathfrak{A}} = \mathbf{f}; \\ \bot, & \text{otherwise;} \end{cases}$$

$$\left[\!\!\left[\begin{matrix} \textbf{if } \phi \textbf{ then } \psi \\ \textbf{else } \chi \textbf{ fi} \end{matrix} \right]\!\!\right]^{\mathfrak{A}} = \begin{cases} \bot, & \text{if } [\![\phi]\!]^{\mathfrak{A}} = \bot; \\ [\![\psi]\!]^{\mathfrak{A}}, & \text{if } [\![\phi]\!]^{\mathfrak{A}} = \mathbf{t}; \\ [\![\chi]\!]^{\mathfrak{A}}, & \text{if } [\![\phi]\!]^{\mathfrak{A}} = \mathbf{f}. \end{cases}$$

We say that $\mathfrak{A}$ *satisfies* a formula ϕ if $[\![\phi]\!]^{\mathfrak{A}} = \mathbf{t}$. We say that $\mathfrak{A}$ *satisfies* a theory T if it satisfies all formulae of T.

According to the Definition 5, undefinedness arises only by application of a partial function to entities outside its domain. If all functions are total, the truth assignment coincides with the standard two-valued assignment of classical first-order logic. Similarly, if a formula is well-defined in a partial function structure according to the three-valued semantics, it preserves the value in the two-valued semantics in a structure obtained by expanding all partial functions to an arbitrary value outside of their domain. Towards the formalization of this property, we define such an expansion of a partial function structure.

Definition 6 A *total expansion* of a partial function structure $\mathfrak{A}$ is a structure $\mathfrak{A}^{\uparrow}$ with the same universe as $\mathfrak{A}$ such that, for each predicate symbol $p \in \Sigma$ (including domain and graph predicates), $p^{\mathfrak{A}^{\uparrow}}$ equals $p^{\mathfrak{A}}$, and for each function symbol $f \in \Sigma$, $f^{\mathfrak{A}^{\uparrow}}$ is a total function expanding the partial function $f^{\mathfrak{A}}$, i.e., $f^{\mathfrak{A}} \subseteq f^{\mathfrak{A}^{\uparrow}}$. This function coincides with $f^{\mathfrak{A}}$ on elements in the domain of $f^{\mathfrak{A}}$ and otherwise assigns an arbitrary element of the universe.

Remark 1 The total expansion of a partial function structure preserves information about the partiality of functions via the domain and graph predicates. This property is important for Theorem 1.

The following theorem shows that an FO(PF) expression α that is well-defined in a partial function structure $\mathfrak{A}$ (i.e., $[\![\alpha]\!]^{\mathfrak{A}} \neq \bot$) has the same value in any total expansion $\mathfrak{A}^{\uparrow}$ of $\mathfrak{A}$.

We use the notation $\|\alpha\|^{\mathfrak{B}}$ to refer to the value of an expression α in an FO structure $\mathfrak{B}$ under the standard two-valued semantics.

Theorem 1 *Let α be an FO(PF) expression over vocabulary Σ, $\mathfrak{A}$ a structure (over Σ), and $\mathfrak{A}^{\uparrow}$ an arbitrary total expansion of $\mathfrak{A}$. Then $[\![\alpha]\!]^{\mathfrak{A}} \neq \bot$ implies $[\![\alpha]\!]^{\mathfrak{A}} = \|\alpha\|^{\mathfrak{A}^{\uparrow}}$.*

Proof We prove that $[\![\alpha]\!]^{\mathfrak{A}} \neq \bot$ implies $[\![\alpha]\!]^{\mathfrak{A}} = \|\alpha\|^{\mathfrak{A}^{\uparrow}}$ by structural induction on α. Let us consider the case where α is a term of the form $f(t_1, \ldots, t_n)$. Assuming that $[\![f(t_1, \ldots, t_n)]\!]^{\mathfrak{A}} \neq \bot$, we infer from Definition 5 that $[\![t_i]\!]^{\mathfrak{A}} \neq \bot$ for all i, and $([\![t_1]\!]^{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathfrak{A}}) \in \delta_f^{\mathfrak{A}}$. By the induction hypothesis, the former implies that

$$[\![t_i]\!]^{\mathfrak{A}} = \|t_i\|^{\mathfrak{A}^{\uparrow}}$$

for all i. By Definition 6, the latter implies that

$$f^{\mathfrak{A}}([\![t_1]\!]^{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathfrak{A}}) = f^{\mathfrak{A}^{\uparrow}}([\![t_1]\!]^{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathfrak{A}}).$$

Combining the two, we find that

$$\begin{aligned}
[\![f(t_1, \ldots, t_n)]\!]^{\mathfrak{A}} &= f^{\mathfrak{A}}([\![t_1]\!]^{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathfrak{A}}) \\
&= f^{\mathfrak{A}^{\uparrow}}([\![t_1]\!]^{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathfrak{A}}) \\
&= f^{\mathfrak{A}^{\uparrow}}(\|t_1\|^{\mathfrak{A}^{\uparrow}}, \ldots, \|t_n\|^{\mathfrak{A}^{\uparrow}}) \\
&= \|f(t_1, \ldots, t_n)\|^{\mathfrak{A}^{\uparrow}}.
\end{aligned}$$

The proofs for the other cases are straightforward. □

Kleene [6, p. 334] called this property the *regularity* of language connectives and expressed it as a property of truth tables. Fitting [8] explored it as a *monotonicity* property of language connectives in an order where $\bot$ is below both $\mathbf{t}$ and $\mathbf{f}$.

5 Guarded FO(PF)

In this section, we propose syntactical constraints to ensure the well-definedness of an FO(PF) formula. The building blocks of such formulae are *guards* (hence the name Guarded FO(PF)). These *guards* ensure that partial function terms occur only in subformulae where it is guaranteed that these terms denote.

For example, to "guard" the atom $1/x > 0$ against division by 0, one can use **if** $\delta_/(1, x)$ **then** $1/x > 0$ **else** ϕ **fi** where ϕ can be $\mathbf{t}$ or $\mathbf{f}$ (or another guarded formula) depending on the desired outcome when $x = 0$. We call such guarding a *conditional guarding*. In this example, the subformula $1/x > 0$ is guarded because it appears in the *context* of the domain predicate $\delta_/(1, x)$. In the simplest form of guarding, for a formula ψ to be in the *context* of a domain predicate $\delta_f(\bar{t})$ means that ψ is in the "then" part of an *if-then-else* statement where $\delta_f(\bar{t})$ is in the condition, i.e., **if** $\delta_f(\bar{t})$ **then** ψ **else** ϕ **fi**. This entails that the partial function term $f(\bar{t})$ can safely occur in formula ψ, as we know that $\bar{t}$ is in the domain of f. Formally, we define a *guarding relation* $\Vdash$ between *guarding contexts* ω, which are sets of domain predicate atoms $\delta_f(\bar{t})$, and FO(PF) expressions α, by letting $\omega \Vdash \alpha$ iff for every function term $f(\bar{t})$ in α, the corresponding domain predicate atom $\delta_f(\bar{t})$ is in ω, or $f(\bar{t})$ occurs in the "then" part of a conditional whose condition contains $\delta_f(\bar{t})$. It should be intuitively clear that a guarded formula cannot be undefined, in the sense that it always evaluates to true or false, as made precise in Theorem 2.

Definition 7 Given a vocabulary Σ, a *guarding context* ω (over Σ) is a set of domain predicate atoms over vocabulary Σ.

The guarding relation $\Vdash$ between guarding contexts ω and FO(PF) expressions α is defined by structural induction on α. The definition is presented as a set of inference rules (also known as typing judgments in type systems literature). Such inference rules are of the form $\dfrac{\psi_1 \dots \psi_n}{\phi}\,N$, which should be read as "if the premises $\psi_1 \dots \psi_n$ hold, then the conclusion ϕ holds". Here, N is the name of the rule.

Definition 8 Let Σ be a vocabulary. The *guarding relation* $\Vdash$ is a binary relation between guarding contexts ω (over Σ) and FO(PF) expressions α (over Σ). It is defined by structural induction on α via the following rules:

$$\frac{}{\omega \Vdash \mathbf{t}}\,G_{tr} \qquad \frac{}{\omega \Vdash \mathbf{f}}\,G_{fa} \qquad \frac{}{\omega \Vdash x}\,G_{var} \qquad \frac{\omega \Vdash \bar{t}}{\omega \Vdash p(\bar{t})}\,G_{atom} \qquad \frac{\omega \Vdash \phi}{\omega \Vdash \neg\phi}\,G_{neg}$$

$$\frac{\omega \Vdash \phi \quad \omega \Vdash \psi}{\omega \Vdash \phi \lor \psi}\,G_{dis} \qquad \frac{\omega \Vdash \phi}{\omega \Vdash \exists x : \phi}\,G_{eq} \qquad \frac{\delta_f(\bar{t}) \in \omega \quad \omega \Vdash \bar{t}}{\omega \Vdash f(\bar{t})}\,G_{term}$$

$$\frac{\omega \Vdash \bar{t_1} \dots \omega \Vdash \bar{t_n} \quad \omega \Vdash \phi \quad \omega \cup \{\delta_{f_1}(\bar{t_1}), \dots, \delta_{f_n}(\bar{t_n})\} \Vdash \psi \quad \omega \Vdash \chi}{\omega \Vdash \mathbf{if}\ \delta_{f_1}(\bar{t_1}) \land \dots \land \delta_{f_n}(\bar{t_n}) \land \phi\ \mathbf{then}\ \psi\ \mathbf{else}\ \chi\ \mathbf{fi}}\,G_{cond}$$

Here $\bar{t}$ stands for the terms $t_1, \dots, t_n$, and $\omega \Vdash \bar{t}$ stands for the premises $\omega \Vdash t_1 \dots\ \omega \Vdash t_n$ (or for no premises if $n = 0$). If $\omega \Vdash \alpha$, we say that α is *guarded* in the context ω. If α is guarded in the empty context $\emptyset$, we say that α is *well-guarded.*

In many situations, there are function symbols that are always total. In integer arithmetic, for instance, all numerals $(\dots, -1, 0, 1, \dots)$ denote, and so do the addition $(+)$, subtraction $(-)$, and multiplication $(\times)$ of any two numbers. For this reason, it is convenient to differentiate between total and partial functions at the vocabulary level. In practical KR systems supporting partial functions, users may specify which of their function symbols represent total functions and which represent partial functions. For built-in functions (like arithmetic operators), this is specified by the designers of the systems.

In our formalism, we can take these total function symbols into account by including them in our guarding context. Given a vocabulary Σ, we define a guarding context ω_{tc} consisting of all domain atoms δ_c for which c is a total constant symbol. Correspondingly, each total constant symbol in Σ is assigned a value in each partial function structure $\mathfrak{A}$. To additionally take non-constant total function symbols into account, we need an extension of our notion of guarding context, as will be discussed in Section 5.4. Finally, we say that an FO(PF) expression α over vocabulary Σ is *well-guarded* in Σ if $\omega_{tc} \Vdash \alpha$.

Example 1 The formula $\exists x : 10/x = 1$ is not well-guarded (although it is well-defined as commonly interpreted in the natural numbers), as the term $10/x$ is not guarded. The guarded version of the statement, under the assumption that the constants 10 and 1 are total, is:

$$\exists x : \textbf{if } \delta_{/}(10, x) \textbf{ then } 10/x = 1 \textbf{ else f fi}$$

We show this by deriving that this formula is well-guarded in the context ω_{tc} containing δ_n for each natural number n (meaning that symbols n denote). For compactness, we write ω'_{tc} for $\omega_{tc} \cup \{\delta_{/}(10, x)\}$. This derivation should be read bottom-up.

$$\dfrac{\dfrac{\dfrac{\checkmark}{\delta_{/}(10,x) \in \omega'_{tc}} \quad \dfrac{\dfrac{\checkmark}{\delta_{10} \in \omega'_{tc}}}{\omega'_{tc} \Vdash 10}\,G_{term} \quad \dfrac{\checkmark}{\omega'_{tc} \Vdash x}\,G_{var}}{\omega'_{tc} \Vdash 10/x}\,G_{term} \quad \dfrac{\dfrac{\checkmark}{\delta_1 \in \omega'_{tc}}}{\omega'_{tc} \Vdash 1}\,G_{term}}{\omega'_{tc} \Vdash 10/x = 1}\,G_{atom}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\checkmark}{\delta_{10} \in \omega_{tc}}}{\omega_{tc} \Vdash 10}\,G_{term} \quad \dfrac{\checkmark}{\omega_{tc} \Vdash x}\,G_{var}}{\omega_{tc} \Vdash \delta_{/}(10,x)}\,G_{atom} \quad \vdots \quad \dfrac{\checkmark}{\omega_{tc} \Vdash \textbf{f}}\,G_{fa}}{\omega_{tc} \Vdash \textbf{if } \delta_{/}(10,x) \textbf{ then } 10/x = 1 \textbf{ else f fi}}\,G_{cond}}{\omega_{tc} \Vdash \exists x : \textbf{if } \delta_{/}(10,x) \textbf{ then } 10/x = 1 \textbf{ else f fi}}\,G_{eq}$$

(Recall that $=$ is a binary predicate symbol.)

Finally, the following theorem states the main property of well-guarded expressions always being two-valued.[11] We prove this result for a general context ω. In particular, it holds for the context ω_{tc}.

Theorem 2 *Given an FO(PF) expression α over vocabulary Σ and a guarding context ω, if $\omega \Vdash \alpha$, then $[\![\alpha]\!]^{\mathfrak{A}} \neq \bot$ for every partial function structure $\mathfrak{A}$ over Σ that satisfies ω.* [12]

Proof The proof is by structural induction on α. We discuss the two most interesting cases. Proving the other cases is straightforward.

Case 1: α is of the form $f(\bar{t})$. Suppose that $\omega \Vdash f(\bar{t})$. By inspection of Definition 8, we must have $\delta_f(\bar{t}) \in \omega$ and $\omega \Vdash \bar{t}$. Let $\mathfrak{A}$ be a partial function structure over Σ satisfying ω. By induction hypothesis, $\omega \Vdash \bar{t}$ implies that $[\![t_i]\!]^{\mathfrak{A}} \neq \bot$ for all i. Furthermore, since $\delta_f(\bar{t}) \in \omega$, we have that $\mathfrak{A} \models \delta_f(\bar{t})$. As a consequence, $[\![\alpha]\!]^{\mathfrak{A}} \neq \bot$.

[11] This is what distinguishes our work from all the existing approaches discussed in Section 2.
[12] Note that a context ω is a theory. Hence, the requirement that $\mathfrak{A}$ satisfies ω means that $\mathfrak{A}$ satisfies all formulae in ω.

Case 2: α is of the form **if** ϕ **then** ψ **else** χ **fi**. Suppose that $\omega \Vdash$ **if** ϕ **then** ψ **else** χ **fi**. By inspection of Definition 8, there must be domain atoms $\delta_{f_1}(\bar{t}_1), \ldots, \delta_{f_n}(\bar{t}_n)$ and a formula ϕ' such that ϕ equals $\delta_{f_1}(\bar{t}_1) \wedge \cdots \wedge \delta_{f_n}(\bar{t}_n) \wedge \phi'$, and $\omega \Vdash \bar{t}_1$, ..., $\omega \Vdash \bar{t}_n$, $\omega \Vdash \phi'$, $\omega \cup \{\delta_{f_1}(\bar{t}_1), \ldots, \delta_{f_n}(\bar{t}_n)\} \Vdash \psi$, and $\omega \Vdash \chi$. (Possibly, $n = 0$ and $\phi' = \phi$.) Let $\mathfrak{A}$ be a partial function structure over Σ satisfying ω. By induction hypothesis, $\omega \Vdash \phi'$ implies $[\![\phi']\!]^{\mathfrak{A}} \neq \bot$. Since $[\![\delta_{f_i}(\bar{t}_i)]\!]^{\mathfrak{A}} \neq \bot$ for all i, it follows that $[\![\phi]\!]^{\mathfrak{A}} = [\![\delta_{f_1}(\bar{t}_1) \wedge \cdots \wedge \delta_{f_n}(\bar{t}_n) \wedge \phi']\!]^{\mathfrak{A}} \neq \bot$. Then either $[\![\phi]\!]^{\mathfrak{A}} = \mathbf{t}$ or $[\![\phi]\!]^{\mathfrak{A}} = \mathbf{f}$.

In case $[\![\phi]\!]^{\mathfrak{A}} = \mathbf{t}$, we have that $[\![\delta_{f_i}(\bar{t}_i)]\!]^{\mathfrak{A}} = \mathbf{t}$ for all i. It follows that $\mathfrak{A} \models \omega \cup \{\delta_{f_1}(\bar{t}_1), \ldots, \delta_{f_n}(\bar{t}_n)\}$. Applying the induction hypothesis on the fact that $\omega \cup \{\delta_{f_1}(\bar{t}_1), \ldots, \delta_{f_n}(\bar{t}_n)\} \Vdash \psi$, we find that $[\![\psi]\!]^{\mathfrak{A}} \neq \bot$. Hence $[\![\alpha]\!]^{\mathfrak{A}} = [\![\psi]\!]^{\mathfrak{A}} \neq \bot$.

In case $[\![\phi]\!]^{\mathfrak{A}} = \mathbf{f}$, we have $[\![\alpha]\!]^{\mathfrak{A}} = [\![\chi]\!]^{\mathfrak{A}} \neq \bot$, where the latter follows from $\omega \Vdash \chi$ by induction hypothesis. □

The converse of Theorem 2 does not hold in general. A counterexample is the formula **if** $x \neq 0$ **then** $1/x > 0$ **else f fi**. It is clearly well-defined, but not well-guarded according to Definition 8.

By applying the previous theorem to the context ω_{tc}, we obtain that if $\omega_{tc} \Vdash \alpha$, then $[\![\alpha]\!]^{\mathfrak{A}} \neq \bot$ for every partial function structure $\mathfrak{A}$ over Σ that satisfies ω_{tc}. In words, one could say that every well-guarded expression is well-defined.

Checking well-guardedness of an FO(PF) expression can be done by simply following the rules of Definition 8. This yields a decidable method that runs in polynomial time.

Theorem 3 *The well-guardedness of a given FO(PF) expression in a given context can be checked in polynomial time.*

Proof To check whether a formula is well-guarded in a context, it suffices to traverse its Abstract Syntax Tree (AST) once.[13] The recursive procedure is specified in Algorithm 1.[14] It takes as input a context of domain predicate atoms and the AST of an FO(PF) expression, and it decides whether the expression is guarded in the context. The function *guards* used in this procedure takes as input an FO(PF) formula ψ, and returns an array of all domain atoms occurring in ψ as a top-level conjunct. This function is specified in Algorithm 2. □

5.1 Conjunctive and implicative guarding

According to Definition 8, an FO(PF) formula ϕ can only be well-guarded if each of its terms $f(\bar{t})$ occurs strictly in the "then" part (i.e., within ψ in formula 1) of a conditional subformula of ϕ, where the constraint includes the domain atom $\delta_f(\bar{t})$. That is, ϕ must contain a subformula of the form:

$$\textbf{if } \cdots \wedge \delta_f(\bar{t}) \wedge \ldots \textbf{ then } \psi \textbf{ else } \ldots \textbf{ fi} \tag{1}$$

[13] An AST of an FO(PF) formula is a finite rooted tree that represents the syntactic structure of the formula: each internal node is a logical operator, quantifier, or a predicate/function application; each leaf is a variable, proposition, or constant.

[14] Note that this procedure can be easily adapted to generate a detailed report of unguarded terms, which is particularly useful for reporting errors to the end-user.

Algorithm 1 Procedure to calculate $\omega \Vdash a$

```
Input: $\omega$ – an array of domain atoms over $\Sigma$
Input: $a$ – an abstract syntax tree of an FO(PF) formula over $\Sigma$
Output: $True$ if $\omega \Vdash a$ and $False$ otherwise
1: procedure $\Vdash(\omega, a)$
2:     $n \leftarrow root(a)$                                      ▷ Let $n$ be the root node in AST $a$
3:     if $if\!-\!then\!-\!else(n)$ then                            ▷ $n$ is an if-then-else node
4:         $gc \leftarrow \Vdash(\omega, cond(n))$                  ▷ Check "condition"
5:         $gt \leftarrow \Vdash(\omega \cup guards(cond(n)), then(n))$ ▷ Check "then" part in the context
6:                                                                  extended with all domain atoms
                                                                    from the "condition"
7:         $ge \leftarrow \Vdash(\omega, else(n))$                  ▷ Check else part
8:         return $gc \wedge gt \wedge ge$
9:     else                                                         ▷ $n$ is any other node
10:        $g \leftarrow True$                                      ▷ Let $g$ be true
11:        if $term(n)$ then                                        ▷ If $n$ is a term node
12:            $g \leftarrow var(n) \vee \delta_n \in \omega$       ▷ Check if $n$ is a variable or if the domain
13:                                                                 atom of $n$ is in the context
14:        end if
15:        for $n_s \in sub(n)$ do                 ▷ For each subterms/subformulae $(n_s)$ in $n$
16:            $g \leftarrow g \wedge \Vdash(\omega, n_s)$   ▷ Check subterms/subformulae and save in $g$
17:        end for
18:        return $g$
19:    end if
20: end procedure
```

While this form of guarding provably leads to two-valued formulae in every structure, it also leads to long, verbose formulae that quickly become cumbersome to read and write. Below, we introduce more convenient guarding constructs that continue to ensure well-definedness and non-ambiguity.

In most sensible cases, the *else* part of a conditional (if-then-else) guard is either **t** or **f**. For these cases, we introduce the *directional conjunction* $\overrightarrow{\wedge}$ and the *directional implication* $\overrightarrow{\Rightarrow}$, which are defined as follows:[15]

$$\phi \overrightarrow{\wedge} \psi \doteq \textbf{if } \phi \textbf{ then } \psi \textbf{ else f fi}$$
$$\phi \overrightarrow{\Rightarrow} \psi \doteq \textbf{if } \phi \textbf{ then } \psi \textbf{ else t fi}$$

Here $\doteq$ stands for syntactic equality. The directional conjunction $(\overrightarrow{\wedge})$ satisfies the associativity law, while commutativity is lost. The exportation law $\big((p \overrightarrow{\wedge} q) \overrightarrow{\Rightarrow} r\big) \Leftrightarrow \big(p \overrightarrow{\Rightarrow} (q \overrightarrow{\Rightarrow} r)\big)$ also holds, as can be easily checked.

Using these new connectives, the formula from Example 1 can be expressed as

$$\exists x : \delta_/(10, x) \overrightarrow{\wedge} 10/x = 1.$$

[15]The reader is invited to read McCarthy's left-sequential 3-valued propositional logic [11], and Fitting [8] for more details.

Algorithm 2 Procedure to extract guards from a formula ψ

```
Input: $\psi$ – an FO(PF) formula
Output: Array of domain predicate atoms in $\psi$
procedure $guards(\psi)$
    $n \leftarrow root(\psi)$                          ▷ Let $n$ be the root node in AST $\psi$
    if $atom(n)$ then                                  ▷ $n$ is an atomic formula
        if $delta(n)$ then                             ▷ $n$ is a delta predicate
            return $n$
        end if
    else if $conjunction(n)$ then                      ▷ $n$ is a conjunction
        return $guards(left(n)) \cup guards(right(n))$  ▷ Return guards from
                                                       left and right conjunct
    else                                               ▷ Otherwise
        return $\emptyset$                             ▷ Return an empty array
    end if
end procedure
```

Furthermore, directional connectives allow for an easier chaining of guards. Consider the following example: "Alex's mother is a doctor", modeled as $Doctor(mother(alex))$. Using directional connectives, this formula can be guarded as

$$\delta_{alex} \overrightarrow{\wedge} \delta_{mother}(alex) \overrightarrow{\wedge} Doctor(mother(alex)).$$

Notice that the order of the guards is important. The term *alex* has to be guarded first as it occurs as an argument in the guard for the *mother* function.

We extend the guarding relation with explicit rules for the directional connectives $\overrightarrow{\wedge}$ and $\Rrightarrow$ (which can be derived from their definition in terms of the conditional connective).

Definition 9 The *guarding relation* from Definition 8 is extended with the following rules:

$$\frac{\omega \Vdash \bar{t} \quad \omega \cup \{\delta_f(\bar{t})\} \Vdash \psi}{\omega \Vdash \delta_f(\bar{t}) \overrightarrow{\wedge} \psi}\ G_{con} \qquad \frac{\omega \Vdash \bar{t} \quad \omega \cup \{\delta_f(\bar{t})\} \Vdash \psi}{\omega \Vdash \delta_f(\bar{t}) \Rrightarrow \psi}\ G_{imp}$$

5.2 Term guarding

To guard partial function terms as described above, a domain predicate must be introduced for each subterm. This method becomes cumbersome for already fairly small terms. To mitigate this issue, we introduce a new unary meta-function Δ that sends FO(PF) terms to FO(PF) formulae. Informally, one can see this function as a predicate expressing that a term[16] is well-defined. Formally:

[16] A similar operator is introduced by Gavilanes-Franco and Lucio-Carrasco [9], where Δt evaluates to true if term t is defined and to false otherwise. In their approach, Δ is a total and non-strict operator (i.e., it yields a value even when applied to an undefined term).

Definition 10 Given an FO(PF) term t, we define an FO(PF) formula $\Delta(t)$ by structural induction on t, via the following rules:

$$\Delta(x) \doteq \mathbf{t} \text{ if } t \text{ is a variable } x$$
$$\Delta(f(\bar{s})) \doteq \Delta(\bar{s}) \overrightarrow{\land} \delta_f(\bar{s}) \text{ if } t \text{ is a non-variable term } f(\bar{s})$$

Here we used $\Delta(\bar{s})$ as shorthand for $\Delta(s_1) \overrightarrow{\land} \ldots \overrightarrow{\land} \Delta(s_n)$, where $\bar{s} \doteq s_1, \ldots, s_n$.[17]

For example, the statement "Alex's mother is a doctor" is guarded with Δ as follows:

$$\Delta(mother(alex)) \overrightarrow{\land} Doctor(mother(alex)).$$

This formula is well-guarded as $\Delta(mother(alex))$ is a syntactical abbreviation for $\delta_{alex} \overrightarrow{\land} \delta_{mother}(alex)$.

5.3 Implicit guarding

This extension is motivated by the observation that, in many cases, the same disambiguation applies to all terms within a formula. For instance, in the sentence "Alex's mother is a doctor" one usually assumes implicatures that both "Alex" and "Alex's mother" exist. Another example is: "The queen of France is the wife of the king of France". In this case, one may accept this statement as true even when France has no queen or king, i.e., when seen as a statement about the concept of queen of France. Therefore, we introduce two new language constructs.[18] The annotation $[\![\phi]\!]$ stands for *implicit conjunctive guarding* and the annotation $\langle\!\langle\phi\rangle\!\rangle$ for *implicit implicative guarding*. Informally, when a formula is implicitly guarded, it is extended with guards for all the terms it contains. For example:

$$[\![Doctor(mother(alex))]\!] \doteq \delta_{alex} \overrightarrow{\land} \delta_{mother}(alex) \overrightarrow{\land} Doctor(mother(alex))$$
$$\langle\!\langle qof = wife(kof)\rangle\!\rangle \doteq \delta_{qof} \overrightarrow{\land} \delta_{kof} \overrightarrow{\land} \delta_{wife}(kof) \overrightarrow{\Rightarrow} qof = wife(kof)$$

Informally, these two statements stand for: "Alex exists, they have a mother, and their mother is a doctor.", and "If the queen and the king of France exist, and the king of France has a wife, then the queen of France is his wife."

The intuition behind these two language constructs is simple, however, their formalization is more complex. The complexities are introduced by terms that are already guarded, the scope of variables, and the nesting of implicit guards. In the following

[17]Recall that constant symbols are 0-ary function symbols. Hence, they are covered by the second rule, which in this case reduces to $\Delta(c) \doteq \delta_c$.

[18]In [10], de Nivelle introduces similar operators $[A]B$ and $\langle A\rangle B$ (respectively lazy implication and lazy conjunction) which do not evaluate B if A is false (i.e., if A is false then both $[A]B$ and $\langle A\rangle B$ are false). Furthermore, when A is false, it is not even needed to prove that B is a proposition at all (but it is necessary in case A is a proposition and true). In this sense, it is possible to make a connection to our connectives ($[\![\phi]\!]$ and $\langle\!\langle\phi\rangle\!\rangle$) where ϕ needs to be well-defined only when the introduced conjunction of domain atoms $\delta_{t_1} \overrightarrow{\land} \ldots \overrightarrow{\land} \delta_{t_n}$ is true. An important difference is that operators defined in this paper implicitly introduce the guards, relieving the user from explicitly guarding potentially undefined expressions. For example, by assuming that $\delta_{-}(x, y)$ is defined as $\delta_{-}(x, y) \Leftrightarrow Nat(x) \land Nat(y) \land x \geq y$, the example from the Section 2 $\forall xy : [Nat(x) \land Nat(y)]x \geq y \Rightarrow Nat(x - y)$ can be expressed as $\forall x, y : \langle\!\langle Nat(x - y)\rangle\!\rangle$.

three examples, we explain the expected behavior of the new language constructs in these special cases.

Example 2 Consider the statement "the King of Belgium exists, and if the king of France exists, these two individuals cannot be the same", formalized as:

$$\llbracket \delta_{kof} \overrightarrow{\Rightarrow} kof \neq kob \rrbracket$$

Since *kof* is already guarded, the implicit guard $\llbracket \cdot \rrbracket$ should not take it into account, i.e., this sentence stands for (for the readers' convenience, we label the newly introduced guards by underlining them):

$$\underline{\delta_{kob}} \overrightarrow{\wedge} (\delta_{kof} \overrightarrow{\Rightarrow} kof \neq kob)$$

Example 3 Consider the statement "There is a country colored red" formalized as:

$$\llbracket \exists c : colorOf(c) = red \rrbracket$$

Since the scope of the variable c is limited by the quantifier, it is impossible to bring its domain predicate out of it. Hence, this sentence stands for:

$$\underline{\delta_{red}} \overrightarrow{\wedge} \exists c : colorOf(c) = red$$

Notice that this sentence is not well-guarded yet; term $colorOf(c)$ is not guarded.

Example 4 Consider the statement "Given that France is colored, if it is colored blue, then no country is colored red" formalized as:

$$\llbracket colorOf(fr) = blue \Rightarrow \forall c : \langle\!\langle \neg(colorOf(c) = red) \rangle\!\rangle \rrbracket$$

Similarly to Example 2, implicit guards should not take into account terms that are already guarded. In this case term *red* is guarded by $\langle\!\langle \cdot \rangle\!\rangle$ and hence it should not be considered by $\llbracket \cdot \rrbracket$. This becomes clear by first rewriting the $\langle\!\langle \cdot \rangle\!\rangle$ guard, resulting in:

$$\llbracket colorOf(fr) = blue \Rightarrow \forall c : \underline{\delta_{red} \overrightarrow{\wedge} \delta_{colorOf}(c)} \overrightarrow{\Rightarrow} \neg(colorOf(c) = red) \rrbracket$$

Finally, the meaning of the original sentence is:

$$\begin{aligned}
&\underline{\delta_{fr} \overrightarrow{\wedge} \delta_{blue} \overrightarrow{\wedge} \delta_{colorOf}(fr)} \overrightarrow{\wedge} \\
&\qquad \big(colorOf(fr) = blue \Rightarrow \forall c : \underline{\delta_{red} \overrightarrow{\wedge} \delta_{colorOf}(c)} \overrightarrow{\Rightarrow} \neg(colorOf(c) = red)\big)
\end{aligned}$$

The examples above illustrate that defining implicit guards depends on identifying the set of unguarded terms in a formula. We therefore begin by formally defining these terms.

Definition 11 Let ϕ be a formula and t a term. The term t is said to be *guarded* in ϕ if:

- t is a variable term, or
- t is a non-variable term $f(\bar{k})$, and every occurrence of t in ϕ appears within a subformula ψ of a conditional of the form (**if** $\delta_f(\bar{k})$ **then** ψ **else** χ **fi**) in ϕ.

Otherwise, the term t is said to be *unguarded* in ϕ.

Example 5 In formula $\phi \doteq \delta_{kof} \overrightarrow{\Rightarrow} qof = wife(kof)$, term kof is guarded, and terms qof and $wife(kof)$ are unguarded. This is the case because ϕ stands for **if** δ_{kof} **then** $qof = wife(kof)$ **else t fi**.

Finally, we define the two new implicit guarding constructs. By a slight abuse of notation, we will often write δ_t to denote $\delta_f(\bar{k})$ for a non-variable term $t \doteq f(\bar{k})$ (which reduces to δ_c if t is a constant c).

Definition 12 Let ϕ be an FO(PF) formula, and let $\{t_1, \dots, t_n\}$ be the set of all unguarded[19] terms in ϕ that contain no variables introduced in ϕ. We define the *implicit guards* of ϕ as:

$$\llbracket \phi \rrbracket \doteq \delta_{t_1} \overrightarrow{\land} \dots \overrightarrow{\land} \delta_{t_n} \overrightarrow{\land} \phi \qquad \langle\langle \phi \rangle\rangle \doteq \delta_{t_1} \overrightarrow{\land} \dots \overrightarrow{\land} \delta_{t_n} \overrightarrow{\Rightarrow} \phi$$

such that, if a term t_i is a subterm of a term t_j then $i \leq j$.

Proposition 4 $\llbracket \cdot \rrbracket$ *and* $\langle\langle \cdot \rangle\rangle$ *are dual operators, i.e.,* $\langle\langle \phi \rangle\rangle$ *is equivalent to* $\neg\llbracket \neg\phi \rrbracket$ *for all formulae* ϕ*.*

Proof We start by observing that $\neg($**if** ψ **then** $\neg\phi$ **else f fi**$)$ is equivalent to **if** ψ **then** ϕ **else t fi** for all formulae ψ and ϕ. Stated differently, $\neg(\psi \overrightarrow{\land} \neg\phi)$ is equivalent to $\psi \overrightarrow{\Rightarrow} \phi$ for all formulae ψ and ϕ. Picking $\psi \doteq \delta_{t_1} \overrightarrow{\land} \dots \overrightarrow{\land} \delta_{t_n}$ (for all unguarded terms $\{t_1, \dots, t_n\}$ in ϕ), we obtain that $\langle\langle \phi \rangle\rangle$ is equivalent to $\neg\llbracket \neg\phi \rrbracket$ for all formulae ϕ. $\square$

Definition 13 The *guarding relation* from Definition 9 is extended with the rules:

$$\frac{\omega \cup \{\delta_{t_1}, \dots, \delta_{t_n}\} \Vdash \phi}{\omega \Vdash \llbracket \phi \rrbracket}\ G_{i\text{-}con} \qquad \frac{\omega \cup \{\delta_{t_1}, \dots, \delta_{t_n}\} \Vdash \phi}{\omega \Vdash \langle\langle \phi \rangle\rangle}\ G_{i\text{-}imp}$$

Where $\{t_1, \dots, t_n\}$ is the set of all unguarded terms in ϕ.

Recall the approach from *IDP* [16] and *MiniZinc* [15], where the statement $1/x > 0$ is disambiguated as either $\exists y : \gamma_{div}(1, x, y) \land y > 0$ or $\forall y : \gamma_{div}(1, x, y) \Rightarrow y > 0$. In FO(PF), these are expressed as $\llbracket 1/x > 0 \rrbracket$ and $\langle\langle 1/x > 0 \rangle\rangle$.

Note that existential quantifiers usually go with conjunctive guards (i.e., $\exists \bar{x} : \llbracket \phi \rrbracket$) and universal with implicative guards (i.e., $\forall \bar{x} : \langle\langle \phi \rangle\rangle$). However, the theoretical language of partial functions presented so far in this paper does not include any assumptions. Nevertheless, these observations are useful and will be discussed in Section 5.5.

5.4 Indirect guarding

The guarding methods discussed so far in this paper focus on individual formulae rather than on entire theories (i.e., sets of sentences). Grouping sentences into a theory enhances the modularity and compactness of the formalization, which can, in turn, lead to more efficient guarding, as demonstrated in the following example.

[19]Notice that Definition 12 relies on Definition 11 (guarded terms) to handle implicit guards as well (see Example 4). This works because Definition 11 is well-defined for formulae that do not contain implicit guards. Therefore, implicit guards can always be resolved in a bottom-up manner.

Example 6 Let symbols *fr* and *red* be constant symbols (representing country France and color red) and let the theory T be the set of the following sentences:

$$\delta_{red}. \qquad \delta_{fr}. \qquad Country(fr). \tag{ψ_{1-3}}$$

$$\forall x : Country(x) \Rightarrow \delta_{closest}(x). \tag{ψ_4}$$

$$\forall x : Country(x) \Rightarrow \delta_{colorOf}(x). \tag{ψ_5}$$

$$\forall x : Country(x) \Rightarrow Country(closest(x)). \tag{ψ_6}$$

$$colorOf(closest(fr)) = red. \tag{ψ_7}$$

This theory states that the constant *red* denotes (ψ_1), the constant *fr* denotes a country (ψ_{2-3}), the functions *closest* and *colorOf* are defined for all countries (ψ_{4-5}), the function *closest* maps countries to countries (ψ_6), and the color of the county closest to France is red (ψ_7).

The sentences (ψ_3), (ψ_6), and (ψ_7) contain partial functions. We informally justify why each of them is well-defined. Formula (ψ_3) is well-defined since the term *fr* is defined (ψ_2). To show that formula (ψ_6) is well-defined, we shall show that the term $closest(x)$ is defined. Observe that x is constrained in (ψ_6) to be a country. From (ψ_4) follows that *closest* is defined for all countries. Finally, (ψ_7) contains the terms $colorOf(closest(fr))$ and *red*. Term *red* is clearly defined (ψ_1). On the other side, *fr* denotes a country (ψ_{2-3}), and the function *closest* is defined for countries (ψ_4). Hence, $closest(fr)$ is denoting. From (ψ_6) follows that $closest(fr)$ also denotes a country. Function *colorOf* is defined for countries (ψ_5).

Although the theory from Example 6 is well-defined, the sentences (ψ_3), (ψ_6), and (ψ_7) are not well-guarded. Nevertheless, we demonstrated that the well-definedness of this theory can still be justified through *syntactical* reasoning. This aligns with the core principles of the guarding approach, and thus, we extend the notion of guarding to FO(PF) theories. The reasoning outlined in Example 6 can be summarized as follows:

1. The well-definedness of a term in a formula may be entailed by a domain predicate found elsewhere in the theory.
2. Sometimes, well-definedness of a term is not derivable directly from the theory, but through a series of *modus ponens* applications.

Incorporating these ideas into the existing logic requires an extension of the guarding context and guarding relation:

Definition 14 (Extension of Definition 7) - Given a vocabulary Σ, a *guarding context* ω (over Σ) is a set of formulae over Σ of the following format:

$$\forall \bar{x} : P_1(\bar{s}_1) \land \cdots \land P_n(\bar{s}_n) \Rightarrow P(\bar{k}) \land \phi \tag{2}$$

where $P_1, \ldots, P_n$ and P are predicate symbols and ϕ is an FO(PF) formula. Here we see formulae of the form $P(\bar{t}) \land \phi$, $\forall \bar{x} : P(\bar{t}) \land \phi$ and $P_1(\bar{s}_1) \land \cdots \land P_n(\bar{s}_n) \Rightarrow P(\bar{k}) \land \phi$ as special cases of formula 2, where $k = 0$ and/or $\bar{x}$ is empty.

Thus far, a guarding context was used merely to check whether a given domain atom was a member of it. However, after extending the context with formulae from a theory, we aim to determine whether an atomic fact is implied by the context. We introduce a new symbol $\underline{\in}$ for representing this relation, and define it simultaneously with the guarding relation.

Definition 15 The *guarding relation* from Definition 13 is extended with the following rule defining the $\underline{\in}$ relation:

$$\frac{\forall \bar{x} : P_1(\bar{s}_1) \wedge \cdots \wedge P_m(\bar{s}_m) \Rightarrow P(\bar{r}) \wedge \phi \in \omega \quad P_i(\bar{s}_i)[\bar{x} \to \bar{u}] \mathrel{\underline{\in}} \omega}{P(\bar{r}[\bar{x} \to \bar{u}]) \mathrel{\underline{\in}} \omega}\; G_{\textit{p-imp}}$$

The rule $G_{\textit{p-imp}}$ contains the premise $P_i(\bar{s}_i)[\bar{x} \to \bar{u}] \mathrel{\underline{\in}} \omega$ for every i in $1, \ldots, m$ (or no premises if $m = 0$). Here "$\to$" stands for term substitution, and each symbol with a bar (e.g., $\bar{r}$) stands for a tuple of terms. The arities of $\bar{x}$ and $\bar{u}$ must match, while the arities of the $\bar{s}_i$ may differ.

Definition 16 The rules G_{term} and G_{cond} from Definition 13 are replaced with the following rules:

$$\frac{\delta_f(\bar{t}) \mathrel{\underline{\in}} \omega \quad \omega \Vdash \bar{t}}{\omega \Vdash f(\bar{t})}\; G_{term}$$

$$\frac{\omega \Vdash \bar{t}_1 \ldots \omega \Vdash \bar{t}_n \quad \omega \Vdash \phi \quad \omega \cup \{P_1(\bar{t}_1), \ldots, P_n(\bar{t}_n)\} \Vdash \psi \quad \omega \Vdash \chi}{\omega \Vdash \textbf{if } P_1(\bar{t}_1) \wedge \cdots \wedge P_n(\bar{t}_n) \wedge \phi \textbf{ then } \psi \textbf{ else } \chi \textbf{ fi}}\; G_{cond}$$

Notice that the rule $G_{\textit{p-imp}}$ has $\underline{\in}$ in its premise, which makes it possible to "chain" implications from the context. Terms $\bar{u}$ can be understood as arbitrarily chosen such that, by substituting all variables in the terms $\bar{r}$, the desired terms are obtained. The purpose of the G_{cond} rule is similar to the corresponding one from Definition 8, but it introduces additional atoms (not only domain atoms) in the context.[20]

Example 7 Let $\omega = \{\psi_1, \ldots, \psi_6\}$ from the Example 6. We show that the formula ψ_7 from the same example is well-guarded in ω.

$$G_{p-imp}\;\cfrac{\cfrac{\cfrac{\checkmark}{\psi_4 \in \omega} \quad \cfrac{\cfrac{\checkmark}{\psi_3 \in \omega}}{\mathit{Country}(\mathit{fr}) \mathrel{\underline{\in}} \omega}\; G_{p-imp}}{\delta_{closest}(\mathit{fr}) \mathrel{\underline{\in}} \omega} \quad \cfrac{\cfrac{\checkmark}{\psi_2 \in \omega}}{\delta_{\mathit{fr}} \mathrel{\underline{\in}} \omega}\; G_{p-imp}}{\omega \Vdash \mathit{closest}(\mathit{fr})}\; G_{term}$$

$$G_{term}\;\cfrac{\cfrac{\cfrac{\cfrac{\checkmark}{\psi_5 \in \omega} \quad \cfrac{\cfrac{\checkmark}{\psi_6 \in \omega} \quad \cfrac{\cfrac{\checkmark}{\psi_3 \in \omega}}{\mathit{Country}(\mathit{fr}) \mathrel{\underline{\in}} \omega}\; G_{p-imp}}{\mathit{Country}(\mathit{closest}(\mathit{fr})) \mathrel{\underline{\in}} \omega}\; G_{p-imp}}{\delta_{\mathit{colorOf}}(\mathit{closest}(\mathit{fr})) \mathrel{\underline{\in}} \omega}\; G_{p-imp} \quad \vdots}{\omega \Vdash \mathit{colorOf}(\mathit{closest}(\mathit{fr}))} \quad \cfrac{\cfrac{\cfrac{\checkmark}{\psi_1 \in \omega}}{\delta_{red} \mathrel{\underline{\in}} \omega}\; G_{p-imp}}{\omega \Vdash \mathit{red}}\; G_{term}}{\omega \Vdash \mathit{colorOf}(\mathit{closest}(\mathit{fr})) = \mathit{red}}\; G_{atom}$$

[20] Notice that it is possible to replace the extended guarding from Definition 15 with a proof system, and simply check if the domain predicates of unguarded terms are entailed by the theory. This would oppose the approach presented in this paper, which is focused on decidability and prudence.

Recall the beginning of this section where domain atoms were added to the context (ω_{tc}) expressing that constant symbols are denoting. Similarly, using indirect guarding, we can specify total functions. For example, adding a sentence $\forall x : \delta_{colorOf}(x)$ to the context expresses that function *colorOf* is total. Accordingly, we extend the context ω_{tc} over vocabulary Σ to contain domain atoms δ_c for all total constants c in Σ, and formulae $\forall x_1, \ldots, x_n : \delta_f(x_1, \ldots, x_n)$ for all total functions f/n in Σ. Additionally, the context is extended with all formulae from a theory T satisfying the syntactic format from Definition 14. Formally:

Definition 17 A guarding context over an FO(PF) vocabulary Σ (with total constant and function symbols), and an FO(PF) theory T, denoted with ω^T_{tc}, consists of:

- $\forall x_1, \ldots, x_n : \delta_f(x_1, \ldots, x_n) \in \omega^T_{tc}$ for every total function symbol f of arity n (for constants, i.e., $n = 0$, this is $\delta_f \in \omega^T_{tc}$).
- $\phi \in \omega^T_{tc}$ for every formula ϕ in T that satisfies the syntactic format from Definition 14.

With the extended versions of definitions 7 and 8 and the context ω^T_{tc}, we define what it means for an FO(PF) theory to be *well-guarded*.

Definition 18 An FO(PF) theory T over a vocabulary Σ is *well-guarded* if $\omega^T_{tc} \Vdash \psi$ holds for all sentences ψ in T.

The property of well-guarded formulae being well-defined (Theorem 2) extends to well-guarded theories. However, we first need to precisely define what it means for a theory to be well-defined. Informally, an FO(PF) theory is well-defined if the conjunction of all its sentences is well-defined (where the order of conjuncts is not important).

Definition 19 An FO(PF) theory T over a vocabulary Σ is *well-defined* if for every partial function structure $\mathfrak{A}$ over Σ, it holds that $[\![\bigwedge_{\phi \in T} \phi]\!]^{\mathfrak{A}} \neq \bot$.

Theorem 5 *Every well-guarded FO(PF) theory is well-defined.*

Proof Let us first observe the following:

1. From Definition 8 it follows that the well-guarded property is closed under conjunction, i.e., the conjunction of two well-guarded formulae is a well-guarded formula (note that Definition 8 has no explicit rule for conjunction, but it can be obtained from the disjunction and negation rule in the standard way).
2. Theorem 2 (which says that a well-guarded formula is well-defined) holds for the extended definition of the guarding relation (i.e., Definition 15 and 16). This is because structures taken into account in Theorem 2 have to satisfy the context in which the formula is guarded (recall that this context includes a part of the theory; Definition 18). It is obvious that rule $G_{p\text{-}imp}$ operating on the sentences from the context is sound (i.e.,

they are deriving atoms that are entailed by the context) as they are a special form of *modus ponens* inference.

An FO(PF) theory T is interpreted as a conjunction of all its formulae (Definition 19). From the condition that theory T is well-guarded and the first observation, formula $\bigwedge_{\phi \in T} \phi$ is well-guarded (note that this formula is guarded in the context obtained from the formulas of theory T satisfying conditions from Definition 14). Due to the second observation, Theorem 2 applies, and therefore such a formula is well-defined. $\square$

5.5 Pairing guards with quantifiers

Certain guarding patterns are commonly observed in all examples throughout the paper (as pointed out in Section 5.3). In particular, these patterns are present in the context of quantified formulae. For example, the existential quantifier is more often paired with the conjunctive guards, while the universal quantifier is paired with the implicative.

A reason why an existential quantification is commonly not followed by a weak disambiguation (i.e., $\exists x : \langle\langle \phi \rangle\rangle$) is because it is trivially true in case ϕ contains a non-total function (not in the scope of some other quantification). For instance, the statement $\exists x : \langle\langle \mathit{shoesize}(x) = 40 \rangle\rangle$ is true in case the universe contains an object that has no shoe size, e.g., a chair. A more plausible statement is $\exists x : [[\mathit{shoesize}(x) = 40]]$, which says that there exists an object that has a shoe size equal to 40. Similarly, combining a universal quantifier with a strong disambiguation (i.e., $\forall x : [[\phi]]$) yields a statement that is trivially false in case ϕ contains a non-total function. For instance, the formula $\forall x : [[\mathit{shoesize}(x) < 50]]$ is false in case the universe contains an object that has no shoe size. A more plausible statement is $\forall x : \langle\langle \mathit{shoesize}(x) < 50 \rangle\rangle$ which says that any object with a well-defined shoe size has shoe size less than 50. Nevertheless, the statement $\forall x : [[\mathit{shoesize}(x) < 50]]$ is still sensible, in case one wants to express that every object has a well-defined shoe size and it is less than 50.

For this reason, our system makes no default assumptions (as a difference to systems proposed in [15, 16]). Users should themselves specify how to disambiguate any potentially ambiguous part of a statement. Thereby, prudence is maintained and precision is guaranteed.

6 Unnesting terms

Nested function terms are very common in the modeling of human knowledge, e.g., "Alex's father is a doctor", $\mathit{Doctor}(\mathit{father}(\mathit{alex}))$. We demonstrated in the previous section that sentences containing them can carry an ambiguity. Sometimes, there is a need to eliminate functions (e.g., because the preferred solver does not support functions). This can be achieved by *unnesting*, i.e., introducing new quantified variables and replacing functions with their graphs[21]. A good overview of the technique is available in [32]. The first part of this section demonstrates that different unnesting techniques that are equivalence-preserving in classical logic are no longer equivalence-preserving in FO(PF). This directly impacts the ability to automate the unnesting

[21] Recall that the graph predicate of a function f is denoted by γ_f.

process. Offloading this task to the user would significantly increase the complexity of the modeling task. Therefore, restoring this property is important to enable automation and reduce manual effort, which in turn minimizes the potential for user-introduced errors. In the latter part of this section, we demonstrate how to restore equivalence for certain unnesting techniques in a well-guarded formula.

First, we consider the unnesting of a constant symbol by introducing a new existentially or universally quantified variable.

Example 8 Given the formula $\mathit{Doctor}(\mathit{father}(\mathit{alex}))$, the constant *alex* can be unnested (existentially or universally) as follows:

$$\exists x : \gamma_{alex}(x) \land \mathit{Doctor}(\mathit{father}(x))$$
$$\forall x : \gamma_{alex}(x) \Rightarrow \mathit{Doctor}(\mathit{father}(x))$$

Notice that the two formulae are equivalent in classical two-valued logic because there is exactly one x such that $\gamma_{alex}(x)$. This is not the case when the term *alex* fails to denote. If *alex* is non-denoting, then $\gamma_{alex}(x)$ is *false* for every x, and hence the first statement is *false* and the second is *true*.

Next, we demonstrate the unnesting of a constant symbol in different places of the syntax tree.

Example 9 Consider the formula $\mathbf{t} \lor \mathit{Doctor}(\mathit{father}(\mathit{alex}))$ (where $\mathbf{t}$ stands for *true*), and the two following unnestings of the term *alex*:

$$\mathbf{t} \lor (\exists x : \gamma_{alex}(x) \land \mathit{Doctor}(\mathit{father}(x)))$$
$$\exists x : \gamma_{alex}(x) \land (\mathbf{t} \lor \mathit{Doctor}(\mathit{father}(x)))$$

The first formula preserves the truth value since it is tautologically equivalent to the original formula. However, the second formula is *false* when *alex* is non-denoting.

Deeply nested terms introduce additional complexity, as they can be unnested in different orders.

Example 10 In the formula $\mathit{Doctor}(\mathit{father}(\mathit{alex}))$, it is possible first to unnest the term *alex* and then *father*, or vice versa. Different combinations of quantifiers for different terms (universal for *alex* and existential for *father*) results in the following two formulae:

1. Unnesting the term *alex* with a universal quantifier from the formula $\mathit{Doctor}(\mathit{father}(\mathit{alex}))$ results in: $\forall x : \gamma_{alex}(x) \Rightarrow \mathit{Doctor}(\mathit{father}(x))$. Next, unnesting the term $\mathit{father}(x)$ with a existential quantifier results in:

$$\forall x : \gamma_{alex}(x) \Rightarrow (\exists x_1 : \gamma_{father}(x, x_1) \land \mathit{Doctor}(x_1))$$

2. Unnesting the term $\mathit{father}(\mathit{alex})$ with an existential quantifier from the formula $\mathit{Doctor}(\mathit{father}(\mathit{alex}))$ results in: $\exists x : \gamma_{father}(\mathit{alex}, x) \land \mathit{Doctor}(x)$. Further, unnesting the term *alex* with a universal quantifier results in:

$$\exists x : (\forall x_1 : \gamma_{alex}(x_1) \Rightarrow \gamma_{father}(x_1, x)) \land \mathit{Doctor}(x)$$

These formulae are not equivalent. A counter-example would be a world without doctors and where *alex* does not exist.

In conclusion, as the examples above demonstrate, there are numerous ways to eliminate all nested function terms from a formula. This depends on the choices of the quantifier, depth in the syntax tree, and order for each term. The problem is that in FO(PF), different combinations of unnestings are not equivalence-preserving. However, certain choices yield a safely unnested formula, whereby "safe" means that if the original formula is well-defined, the unnested version will have the same value. For example, applying unnesting only on the level of atoms is a safe approach for the following two reasons:

- The problem of unnesting terms in different places of the syntax tree (i.e., Example 9, the second option) is eliminated.
- The condition that the original formula is well-defined eliminates the problem of different quantifiers from Example 8. This is because the replaced term is either defined, in which case both unnestings are equivalent, or it is undefined but its value is not important and can be substituted arbitrarily, which follows from Theorem 1.

Remark 2 Notice that unnesting at the level of atoms replaces undefined atoms with formulae that evaluate either to *true* or *false* (depending on the quantifier choice). Therefore, unnesting all terms from a formula results in a two-valued formula. As this unnesting is not equivalence-preserving in the case of undefined formulae, it follows that for a given formula that is undefined in some structure, there must be one unnesting that yields a formula that is *true* in the structure and one that is *false*.

Returning to the proposed "safe" method, a major downside is that it introduces a new variable for each occurrence of a term.

Example 11 Consider the following formula: "Alex is a doctor and a pilot." modeled as: $Doctor(alex) \wedge Pilot(alex)$. The safe unnesting of this formula is:

$$\exists x_1 : \gamma_{alex}(x_1) \wedge Doctor(x_1) \wedge \exists x_2 : \gamma_{alex}(x_2) \wedge Pilot(x_2)$$

This example shows that an existential quantifier is introduced per occurrence of the term *alex*. For the reasons explained in Example 9, it is not possible to eliminate term *alex* with a single quantifier (i.e., $\exists x : \gamma_{alex}(x) \wedge Doctor(x) \wedge Pilot(x)$). The increased number of quantifiers can impact the solving efficiency negatively (this is known for QBF [33, 34]). However, an improvement is possible for well-guarded formulae. Recall that the issue with Example 9 arises only when some unnested term is non-denoting. From Theorem 2, it follows that non-denoting terms do not affect the truth value of a well-guarded formula. Therefore, different unnestings of a term in the scope of its guards are expected to preserve the value of the formula. To formalize the described property, we first have to define unnesting. We call the one with existential quantifier *strong*, since it approximates a formula to *false* when an undefined term is unnested, and the other one *weak*, as it approximates it to *true*. Below, $\phi[t \to t']$ stands for the substitution of a term t with term t' in formula ϕ.

Definition 20 Given a formula ϕ containing a non-variable term t such that all variables occurring in t are free in ϕ, we define the *weak* and *strong* unnesting of t in ϕ (respectively) as:

$$w(t, \phi) \doteq \forall x : \gamma_t(x) \Rightarrow \phi[t \to x]$$
$$s(t, \phi) \doteq \exists x : \gamma_t(x) \wedge \phi[t \to x]$$

Here x is a fresh variable (i.e., does not occur in t nor in ϕ), and $\gamma_t(x)$ denotes the graph predicate $\gamma_f(\bar{k}, x)$ if $t \doteq f(\bar{k})$ (which reduces to $\gamma_c(x)$ if t is a constant c).

Notice that unnesting a term removes ambiguity, which is present when the term is not denoting. In this sense, unnesting can be seen as a form of disambiguation. Specifically, weak unnesting corresponds to guarding with directional implication ($\overrightarrow{\Rightarrow}$), and strong unnesting to directional conjunction ($\overrightarrow{\wedge}$). However, we do not regard unnesting as a form of guarding, since it requires the user to introduce new variables and use the graph predicate. Conversely, we also cannot define unnesting in terms of guards for similar reasons.

Example 12 Consider the statement "Alex's mother is a doctor and a pilot." formalized as (ϕ):

$$\phi \doteq \mathit{Doctor}(\mathit{mother}(\mathit{alex})) \wedge \mathit{Pilot}(\mathit{mother}(\mathit{alex}))$$

Then the weak unnesting of the term *alex* in ϕ is:

$$w(\mathit{alex}, \phi) \doteq \forall x : \gamma_{alex}(x) \Rightarrow \mathit{Doctor}(\mathit{mother}(x)) \wedge \mathit{Pilot}(\mathit{mother}(x))$$

The double strong unnesting, first of term *mother*(*alex*) and then of term *alex* looks like:

$$\begin{aligned}
&s(\mathit{alex}, s(\mathit{mother}(\mathit{alex}), \phi)) \\
&\doteq s(\mathit{alex}, \exists x : \gamma_{mother}(\mathit{alex}, x) \wedge \mathit{Doctor}(x) \wedge \mathit{Pilot}(x)) \\
&\doteq \exists y : \gamma_{alex}(y) \wedge \exists x : \gamma_{mother}(y, x) \wedge \mathit{Doctor}(x) \wedge \mathit{Pilot}(x)
\end{aligned}$$

Finally, we observe that unnesting a term t inside the scope of a guard of t does not influence the truth value of a statement.

Example 13 Consider the term $t \doteq \mathit{alex}$, the formula $\phi \doteq \mathit{Doctor}(\mathit{alex})$, and an arbitrary formula ψ. Then we have that

$$\begin{aligned}
&[\![\textbf{if } \delta_t \textbf{ then } \phi \textbf{ else } \psi \textbf{ fi}]\!]^{\mathfrak{A}} \\
&= [\![\textbf{if } \delta_t \textbf{ then } w(t, \phi) \textbf{ else } \psi \textbf{ fi}]\!]^{\mathfrak{A}} \\
&= [\![\textbf{if } \delta_t \textbf{ then } s(t, \phi) \textbf{ else } \psi \textbf{ fi}]\!]^{\mathfrak{A}}.
\end{aligned}$$

Indeed, if *alex* does not denote in $\mathfrak{A}$, then all three expressions reduce to $[\![\psi]\!]^{\mathfrak{A}}$. Otherwise, if *alex* does denote in $\mathfrak{A}$, then the equations reduce to

$$[\![\mathit{Doctor}(\mathit{alex})]\!]^{\mathfrak{A}} = [\![\forall x : \gamma_{alex}(x) \Rightarrow \mathit{Doctor}(x)]\!]^{\mathfrak{A}} = [\![\exists x : \gamma_{alex}(x) \wedge \mathit{Doctor}(x)]\!]^{\mathfrak{A}}.$$

Since in this case, $\mathfrak{A}$ contains a single object x such that $\gamma_{alex}(x)$ holds, namely $[\![\mathit{alex}]\!]^{\mathfrak{A}}$, all three expressions are equal.

The following theorem formalizes this property.

Theorem 6 *Let ϕ, ψ, and χ be formulae, t a non-variable term, and $\mathfrak{A}$ a partial function structure (all over the same vocabulary). Then*

$$\begin{aligned} &[\![\textbf{\textit{if}}\ \delta_t\ \textbf{\textit{then}}\ \phi\ \textbf{\textit{else}}\ \psi\ \textbf{\textit{fi}}]\!]^{\mathfrak{A}} \\ =\ &[\![\textbf{\textit{if}}\ \delta_t\ \textbf{\textit{then}}\ \phi[\chi \to w(t,\chi)]\ \textbf{\textit{else}}\ \psi\ \textbf{\textit{fi}}]\!]^{\mathfrak{A}} \\ =\ &[\![\textbf{\textit{if}}\ \delta_t\ \textbf{\textit{then}}\ \phi[\chi \to s(t,\chi)]\ \textbf{\textit{else}}\ \psi\ \textbf{\textit{fi}}]\!]^{\mathfrak{A}}. \end{aligned}$$

Proof If $[\![\delta_t]\!]^{\mathfrak{A}} = \bot$, then all three expressions evaluate to $\bot$ (by Definition 5). If $[\![\delta_t]\!]^{\mathfrak{A}} = \mathbf{f}$, then all three expressions evaluate to $[\![\psi]\!]^{\mathfrak{A}}$ (again by Definition 5). Hence we can assume that $[\![\delta_t]\!]^{\mathfrak{A}} = \mathbf{t}$, which means that t denotes in $\mathfrak{A}$. This implies that

$$[\![\chi]\!]^{\mathfrak{A}} = [\![\forall x : \gamma_t(x) \Rightarrow \chi[t \to x]]\!]^{\mathfrak{A}} = [\![\exists x : \gamma_t(x) \land \chi[t \to x]]\!]^{\mathfrak{A}},$$

as $\mathfrak{A}$ contains a single object x for which $\gamma_t(x)$ is true, namely $[\![t]\!]^{\mathfrak{A}}$. It follows that

$$[\![\phi]\!]^{\mathfrak{A}} = [\![\phi[\chi \to w(t,\chi)]]\!]^{\mathfrak{A}} = [\![\phi[\chi \to s(t,\chi)]]\!]^{\mathfrak{A}},$$

which implies that all three expressions are equal. □

We say that a term t is untested inside the scope of its guards if it is untested as in Theorem 6.

Corollary 6.1 *In a well-guarded FO(PF) formula, any unnesting of a function term inside the scope of its guards is a value-preserving operation.*

The main motivation in this paper for introducing guards was to create a language capable of disambiguating the meaning of expressions using partial functions. We have shown that the well-definedness property is not enough (e.g., $\mathbf{t} \lor c = 10$) to ensure non-ambiguity. However, guards are providing both: well-definedness is reflected in Theorem 2 and non-ambiguity in Theorem 6.

7 Definitions of partial functions

Thus far, we have focused on the analysis of given partial functions. However, from a KR perspective, it is also useful to let users define partial functions. The approach introduced here extends the one of FO(ID), in which inductive definitions of sets are added to classical first-order logic [35, 36]. From a set-theoretic perspective, functions are a special kind of sets. We will translate definitions of functions to definitions of sets via their graph and domain predicates, and rely on FO(ID) to specify their semantics.

7.1 FO(ID)

The content of this section is similar to parts of [37].

In FO(ID), definitions are sets of *definitional rules*. These are of the form

$$\forall \bar{x} : P(\bar{t}) \leftarrow \phi$$

where $\bar{x}$ is a tuple of variables, $P(\bar{t})$ an atom called the *head* of the rule and ϕ an FO formula called the *body* of the rule. Rules of the form $\forall \bar{x} : P(\bar{t}) \leftarrow \mathbf{t}$ are commonly abbreviated to $\forall \bar{x} : P(\bar{t})$. The definitional rules express how to construct the defined predicates, and syntactically resemble the 'if'-sentences commonly used in natural language specifications of inductive definitions. A simple example of a definition in FO(ID) is the following:

$$\left\{ \begin{array}{l} N(0) \\ \forall n : N(s(n)) \leftarrow N(n) \end{array} \right\}$$

It defines N as the set of natural numbers by means of two rules. The first rule can be read as "0 is a natural number", and the second rule as "if n is a natural number, then so is its successor $s(n)$". The predicate symbols that appear in the head of a definitional rule of a definition Φ are called the *defined predicates* of Φ. All other non-logical symbols that appear in Φ are called the *parameters* of Φ. For instance, the definition above has one defined predicate N and two parameters 0 and s.

Extending a logic with definitions makes it (significantly) more expressive. For example, in FO it is impossible to express that a relation $T/2$ is the transitive closure of a relation $E/2$. However, it can be easily represented in FO(ID) by the following definition:

$$\left\{ \begin{array}{l} \forall x, y : T(x,y) \leftarrow E(x,y) \\ \forall x, y, z : T(x,z) \leftarrow T(x,y) \wedge T(y,z) \end{array} \right\} \tag{3}$$

FO(ID) allows for the expression of a wide range of definitions. Many frameworks put a *stratifiability* restriction on their inductive definitions [38–40]. This means that any defined predicate P of a definition can be assigned a *level* $l(P) \in \mathbb{N}$ such that for every predicate symbol Q appearing in the body of a rule defining P, $l(Q) \leq l(P)$, and moreover $l(Q) < l(P)$ if Q occurs negatively in the body. For instance, the definition in (3) is stratifiable, as one can simply pick $l(E) = l(T) = 0$. However, many important definitions are not stratifiable. For example, consider the following FO(ID) definition, which simulates the definition of the satisfiability relation for propositional logic:

$$\left\{ \begin{array}{l} \forall i, p : \mathit{Sat}(i,p) \leftarrow \mathit{In}(p,i) \\ \forall i, f, g : \mathit{Sat}(i, \mathit{and}(f,g)) \leftarrow \mathit{Sat}(i,f) \wedge \mathit{Sat}(i,g) \\ \forall i, f : \mathit{Sat}(i, \mathit{not}(f)) \leftarrow \neg \mathit{Sat}(i,f) \end{array} \right\}$$

Here, a predicate $\mathit{Sat}/2$ is inductively defined in terms of another predicate $\mathit{In}/2$, as well as two constructor functions $\mathit{and}/2$ and $\mathit{not}/1$. The variable i represents a structure, p a propositional constant, and f and g propositional formulas. The term $\mathit{and}(f,g)$ represents the conjunction of f and g, and $\mathit{not}(f)$ the negation of f. This definition is not stratifiable, as it would require a level $l(\mathit{Sat}) \in \mathbb{N}$ for which $l(\mathit{Sat}) < l(\mathit{Sat})$. Therefore, this natural inductive definition cannot be represented in common logics for inductive definitions [38–40].

The fact that definitions in FO(ID) need not be stratified, has as a consequence that they can also be *non-total*. This means that the induction process fails to derive a two-valued interpretation for all defined predicates, and usually indicates a kind of flaw in the definition. A simple example of a non-total definition is

$$\{ P \leftarrow \neg P \} . \tag{4}$$

Non-total definitions are interesting, as they can be linked to paradoxes. For instance, (4) can be seen as a formalization of the well-known liar paradox: "This sentence is false", where P represents the truth value of the sentence. By its definition, it is true precisely if it is false.

The semantics of FO(ID) is inspired by the well-founded semantics of logic programming [41]. We will restrict our discussion of the semantics of FO(ID) to the main ideas in this section, and provide the details in Appendix A. *Well-founded inductions* are mathematical descriptions of the construction processes associated with inductive definitions. More concretely, they take the form of (possibly transfinite) sequences of increasingly precise three-valued structures. These structures are *three-valued* in the sense that they can assign three different truth values to sentences: true, false, and unknown. Unknown is less precise than both true and false, and this preciseness order extends pointwise to three-valued structures. A well-founded induction is defined with respect to a given definition Φ and in a Φ-*context* $\mathcal{O}$, which is a structure interpreting the parameters of Φ. The definitional rules of Φ determine how the three-valued structures can be refined to more precise three-valued structures. Eventually, a well-founded induction reaches a three-valued structure that cannot be further refined. This three-valued structure is called the *well-founded model* of Φ in $\mathcal{O}$. In general, a definition can admit multiple well-founded inductions in a given context, but it has been proven that every such well-founded induction converges to the same three-valued structure [42]. This guarantees that the notion of a well-founded model of Φ in $\mathcal{O}$ is well-defined. Using the notion of well-founded model, we extend the satisfaction relation of FO to FO(ID), by saying that a (two-valued) structure $\mathfrak{A}$ satisfies a definition Φ, $\mathfrak{A} \models \Phi$, if $\mathfrak{A}$ interprets the defined predicates of Φ in the same way as the well-founded model of Φ in the context obtained by restricting $\mathfrak{A}$ to the parameters of Φ.

Note that FO(ID) offers a unique perspective on definitions by treating them as propositions. A definition being true in a structure informally means that the values of its defined concepts correspond to the values derived by the induction process, based on the values of its parameters. The treatment of definitions as propositions permits an analysis of definitions *themselves*, instead of only their defined concepts. The semantics of FO(ID) give an account of how a set of definitional rules (may) give rise to values for defined concepts as the result of an induction process. It furthermore provides explanations of what goes wrong in case of non-totality. FO(ID) thus allows for an investigation of what separates a 'good' (i.e., total) definition from a 'bad' (i.e., non-total) one.

7.2 FO(RD)

To incorporate definitions of partial functions in the framework of FO(ID), we generalize the notion of definitional rule so that a head can also take the form $f(\bar{t}) = k$. The resulting logic is called FO(RD).[22] It is an extension of both FO(PF) and FO(ID).

7.2.1 Syntax

Definition 21 A *definitional rule* is an expression of the form $\forall \bar{x} : H \leftarrow \phi$, where $\bar{x}$ is a sequence of variables, ϕ is an FO(PF) formula called the *body* of the rule, and H is either an atom $P(\bar{t})$ or an equality $f(\bar{t}) = k$ (where f is a partial function symbol and $\bar{t}, k$ are terms), and is called the *head* of the rule. Rules of the form $\forall \bar{x} : H \leftarrow \mathbf{t}$ are abbreviated to $\forall \bar{x} : H$.

Definition 22 A *definition* Φ is a set of definitional rules. The predicate and function symbols that appear in the head of a rule of a definition Φ are called the *defined* predicate and function symbols of Φ, respectively. All other non-logical symbols appearing in Φ are called the *parameters* of Φ. We say that a definition defines its defined (predicate and/or function) symbols in terms of its parameters.

A *formula* in FO(RD) is defined as in FO(PF), but with addition of the following clause:

- If Φ is a definition, then Φ is a formula.

Definition 23 We extend the guarding relation from Definition 8 to definitional rules and definitions, by means of the following rules:

$$\frac{\omega \Vdash \phi}{\omega \Vdash \forall \bar{x} : P(\bar{t}) \leftarrow \Delta(\bar{t}) \land \phi} \quad \frac{\omega \Vdash \phi}{\omega \Vdash \forall \bar{x} : f(\bar{t}) = k \leftarrow \Delta(\bar{t}) \land \Delta(k) \land \phi} \quad \frac{\omega \Vdash r \quad (r \in \Phi)}{\omega \Vdash \Phi}$$

The left rule says that a definitional rule with head of the form $P(\bar{t})$ is guarded in context ω if its body is of the form $\Delta(\bar{t}) \land \phi$, and ϕ is guarded in ω. The middle rule says something similar for definitional rules with head of the form $f(\bar{t}) = k$. The right rule says that a definition is guarded in context ω if all of its definitional rules are guarded in ω.

By Definition 23, an FO(RD) definition is well-guarded if all of its definitional rules have well-guarded bodies that start with a conjunction of atoms $\Delta(t)$ for terms t appearing in the heads of the rules.[23] Recall that $\Delta(t)$ expresses that the term t denotes. We will often simplify bodies of definitional rules by leaving out atoms $\Delta(t)$ which we know evaluate to $\mathbf{t}$, e.g., if t is a variable or a total constant symbol.

Note that the definitional rule $\forall \bar{x} : f(\bar{t}) = k \leftarrow \Delta(\bar{t}) \land \Delta(k) \land \phi$ in the middle of Definition 23 has no atom $\Delta(f(\bar{t}))$ in its body, even though the term $f(\bar{t})$ appears in its head. This is because $f(\bar{t})$ is precisely the term that is (possibly) being constructed by the rule. Strengthening the body to include the atom $\Delta(f(\bar{t}))$ would mean that

[22]The "RD" stands for "recursive definitions". We use the word "recursive" when speaking of definitions of functions, and "inductive" when speaking of definitions of sets.

[23]A similar approach is presented in [5] by Suppes, where division is defined in the following way: If $y \neq 0$, then $x/y = z$ if and only if $x = y \times z$.

one could only define $f(\bar{t})$ in case $f(\bar{t})$ is already defined, which would only result in the construction of trivial functions with empty domain.

One could simplify syntax by removing the restriction that bodies of well-guarded definitional rules should start with atoms $\Delta(t)$, if one follows the convention that if a term in the head does not denote, nothing is derived. However, in our base language, we prefer to remain prudent, forcing the user to be explicit about the use of possibly non-denoting terms.

The following examples illustrate how definitions in FO(RD) provide a versatile tool for naturally specifying various kinds of partial functions.

Example 14 Consider the (partial) function *half* : $\mathbb{N} \to \mathbb{N}$ that sends every even number $2n$ to n and is undefined on every odd number. This function can be defined in FO(RD) in terms of the successor function $s/1$ and the constant 0:

$$\left\{ \begin{array}{l} \mathit{half}(0) = 0 \\ \forall n : \mathit{half}(s(s(n))) = s(\mathit{half}(n)) \leftarrow \Delta(s(\mathit{half}(n))) \end{array} \right\}$$

The first rule assigns value 0 to $\mathit{half}(0)$, and the second rule assigns value $s(\mathit{half}(n))$ to $\mathit{half}(s(s(n)))$ if $s(\mathit{half}(n))$ denotes. Here we simplified the bodies of the rules by using the fact that 0 is a total constant symbol and s a total function symbol.

Example 15 Consider the predecessor function $p : \mathbb{N} \to \mathbb{N}$ that sends every non-zero natural number n to its predecessor $n - 1$ and that is undefined on 0. This function can be defined in FO(RD) in terms of the successor function $s/1$:

$$\left\{ \forall n : p(s(n)) = n \right\}$$

Example 16 Consider the function *minus* : $\mathbb{N}^2 \to \mathbb{N}$ that sends every pair $(n, m) \in \mathbb{N}^2$ with $n \geq m$ to $n - m$ and that is undefined on all other pairs. This function can be defined in FO(RD) in terms of the successor function $s/1$, the constant 0, and the predecessor function $p/1$:

$$\left\{ \begin{array}{l} \forall n : \mathit{minus}(n, 0) = n \\ \forall n, m : \mathit{minus}(n, s(m)) = p(\mathit{minus}(n, m)) \leftarrow \Delta(p(\mathit{minus}(n, m))) \end{array} \right\}$$

Example 17 Consider the function *div* : $\mathbb{N}^2 \to \mathbb{N}$ that sends every pair $(n, m) \in \mathbb{N}^2$ with $m \mid n$ to n/m and that is undefined on all other pairs. This function can be defined in FO(RD) in terms of the successor function $s/1$, the constant 0, and the plus function $\mathit{plus}/2$:

$$\left\{ \begin{array}{l} \forall n : \mathit{div}(n, n) = s(0) \leftarrow n \neq 0 \\ \forall n, m : \mathit{div}(\mathit{plus}(n, m), m) = s(\mathit{div}(n, m)) \leftarrow \Delta(\mathit{div}(n, m)) \end{array} \right\}$$

Example 18 The four elementary arithmetic operations can be defined mutually in FO(RD), in terms of the successor function $s/1$ and the constant 0:

$$\left\{ \begin{array}{l} \forall n : \mathit{plus}(n, 0) = n \\ \forall n, m : \mathit{plus}(n, s(m)) = s(\mathit{plus}(n, m)) \leftarrow \Delta(s(\mathit{plus}(n, m))) \\ \forall n, m, p : \mathit{minus}(n, m) = p \leftarrow [\![n = \mathit{plus}(m, p)]\!] \\ \forall n : \mathit{times}(n, 0) = 0 \\ \forall n, m : \mathit{times}(n, s(m)) = \mathit{plus}(\mathit{times}(n, m), n) \leftarrow \Delta(\mathit{plus}(\mathit{times}(n, m), n)) \\ \forall n, m, p : \mathit{div}(n, m) = p \leftarrow [\![n = \mathit{times}(m, p)]\!] \end{array} \right\}$$

Recall that $[\![n = plus(m,p)]\!]$ stands for $\delta_{plus}(m,p) \overrightarrow{\wedge} (n = plus(m,p))$, which stands for **if** $\delta_{plus}(m,p)$ **then** $n = plus(m,p)$ **else f fi**.

Example 19 Consider the function *element*/2 that is defined precisely on the pairs (i,l) of a natural number i and a list l of length greater than i, for which it returns the i-th element of l (counting from 0). This function can be defined in FO(RD) as follows:

$$\left\{ \begin{array}{l} \forall h,t : element(0,c(h,t)) = h \leftarrow \Delta(c(h,t)) \\ \forall i,h,t : element(s(i),c(h,t)) = element(i,t) \leftarrow \Delta(c(h,t)) \wedge \Delta(element(i,t)) \end{array} \right\}$$

The parameters of this definition are the constant 0, the successor function $s/1$, and the constructor function $c/2$, which maps every pair (h,t) of an object h and a list t to the list with head h and tail t (and is undefined if t is not a list).

Example 20 Consider the function *max*/1 that is defined on every non-empty list of numbers, for which it returns the maximal element of the list. It can be defined in FO(RD) as follows:

$$\left\{ \begin{array}{l} \forall h : max(c(h,nil)) = h \leftarrow \Delta(c(h,nil)) \\ \forall h,t : max(c(h,t)) = h \leftarrow \Delta(c(h,t)) \wedge [\![max(t) \leq h]\!] \\ \forall h,t : max(c(h,t)) = max(t) \leftarrow \Delta(c(h,t)) \wedge \Delta(max(t)) \wedge [\![h \leq max(t)]\!] \end{array} \right\}$$

The parameters of this definition are the binary less than or equal predicate $\leq$, the empty list constant *nil*, and the constructor function c from Example 19. The body of the third rule could be simplified by removing the conjunct $\Delta(max(t))$, as it is implied by $[\![h \leq max(t)]\!]$.

Example 21 The distance from a node n to a node m in a graph is defined as the length of the shortest path from n to m. Given a predicate $E/2$, representing the edge relation of the graph, one can define the distance function $d/2$ of nodes in FO(RD) as follows:

$$\left\{ \begin{array}{l} \forall n : d(n,n) = 0 \\ \forall n,m,p : d(n,m) = s(d(n,p)) \leftarrow \Delta(s(d(n,p))) \wedge (n \neq m) \wedge E(p,m) \wedge \\ \qquad\qquad\qquad\qquad\qquad \neg\exists q : E(q,m) \wedge [\![d(n,q) < d(n,p)]\!] \end{array} \right\}$$

The second rule says that the distance from a node n to another node m is one more than the distance from n to a node p, if there is an edge from p to m, and there is no other node q that has an edge to m and is closer to n (i.e., $d(n,q) < d(n,p)$). Note that d is partial in general, as there may be pairs of nodes between which no path exists.

Example 22 Given a (symmetric) predicate *Married*/2, one can define a function *spouse*/1 in FO(RD) as follows:

$$\{ \forall x,y : spouse(x) = y \leftarrow Married(x,y) \}$$

The defined function is partial in general, as not every person is married.

Example 23 Consider a scenario in which persons can give scores to movies. This can be modeled by means of a predicate *Score*/3, where an atom $Score(p,m,s)$ expresses that person p gives score s to movie m. Based on this predicate, we can define a function *favorite*/1, where $favorite(p) = m$ means that p assigns a unique highest score to movie m:

$$\left\{ \begin{array}{l} \forall p,m : favorite(p) = m \leftarrow \exists s : Score(p,m,s) \wedge \\ \qquad\qquad\qquad\qquad \neg\exists m',s' : m' \neq m \wedge Score(p,m',s') \wedge s' \geq s \end{array} \right\}$$

This function is partial, as it is only defined for persons (which appear as the first argument in *Score*), and moreover, only for persons that assign a unique highest score to a movie.

We demonstrate how our approach applies to ASP on an example originally proposed by Cabalar [17, Example 1].

Example 24 Consider the following scenario: A regular customer at a restaurant follows a specific rule when choosing their second course. If it is Friday, they choose the same dish for the second course as they had for the first. On any other day, if the first course is pasta, they choose fish for the second course.

We model this example using partial constants *first*/0, and *second*/0 (for the first and the second course of a meal), total constants *pasta*/0 and *fish*/0 (representing dishes), and a propositional symbol *Friday*/0 (which is true iff it is Friday).

$$\left\{ \begin{array}{l} \mathit{second} = \mathit{fish} \leftarrow [\![\mathit{first} = \mathit{pasta}]\!] \land \neg \mathit{Friday} \\ \mathit{second} = \mathit{first} \leftarrow \Delta(\mathit{first}) \land \mathit{Friday} \end{array} \right\}$$

The customer "skips" the second course (i.e., *second* is undefined) in two situations: 1) it is Friday and they did not have the first course, 2) it is not Friday and the first course was not pasta.

Assuming ASP with function symbols extended with constructs Δ and $[\![\cdot]\!]$, the definition above can be translated into an ASP by replacing symbols $=$, $\leftarrow$, $\land$, and $\neg$ respectively with $:=$, $:-$, comma (","), and default negation ("not").

```
second := fish :- [[first = pasta]], not Friday.
second := first :- Δ(first), Friday.
```

In this example, we presented a hypothetical ASP program extended with the language constructs introduced in this paper. In Example 27, we continue with this scenario and demonstrate how to derive an ASP encoding from the definition above. Note that the bodies of rules in FO(RD) are less restricted than those in standard ASP. For instance, ASP does not support explicit quantification in the bodies of rules. Hence, translating definitions like the one from Example 21 is not always as straightforward as the one above.

7.2.2 Semantics

The semantics of FO(RD) is defined via the semantics of FO(ID). More specifically, we define what it means for a partial function structure $\mathfrak{A}$ to *satisfy* an FO(RD) formula ϕ in terms of the satisfaction relation for FO(ID), by translating ϕ to an FO(ID) formula $\tau(\phi)$ and $\mathfrak{A}$ to an FO structure $\tau(\mathfrak{A})$.

To handle the potential nesting of partial functions, we introduce an auxiliary (meta-)function G that sends pairs (t, y) of a term t and a variable y to FO formulae. It is defined by structural induction on t, and translates the partial functions occurring in t to existential statements involving the corresponding graph predicates. Informally, the formula $G(t, y)$ expresses that the term t denotes and is equal to y.

Definition 24 Given a term t and a variable y, we define the FO formula $G(t, y)$ by structural induction on t via the following rules:

- $G(x, y) \doteq (x = y)$

- $G(f(\bar{s}), y) \doteq \exists \bar{x}_{\bar{s}} : G(\bar{s}, \bar{x}_{\bar{s}}) \land \gamma_f(\bar{x}_{\bar{s}}, y)$

Here $\bar{x}_{\bar{s}} \doteq x_{s_1}, \dots, x_{s_n}$ are 'designated' variables that are solely used to refer to the terms $\bar{s} \doteq s_1, \dots, s_n$ respectively (if they denote). They are mutually distinct and differ from all 'regular' variables. The expression $G(\bar{s}, \bar{x}_{\bar{s}})$ is shorthand notation for $\bigwedge_{i=1}^{n} G(s_i, x_{s_i})$. Recall that γ_f denotes the graph predicate corresponding to the (partial) function f. Note that since we view constant symbols as 0-ary function symbols, they are covered by this rule. As a constant symbol c takes no arguments, the rule simplifies to $G(c, y) \doteq \gamma_c(y)$ in this case.

Definition 25 The *translation* $\tau(\phi)$ of a formula (resp. definitional rule) ϕ in FO(RD) is a formula (resp. definitional rule) in FO(ID), defined by structural induction via the following rules:

- $\tau(P(\bar{t})) \doteq \exists \bar{x}_{\bar{t}} : G(\bar{t}, \bar{x}_{\bar{t}}) \land P(\bar{x}_{\bar{t}})$
- $\tau(t = s) \doteq \exists x_t, x_s : G(t, x_t) \land G(s, x_s) \land x_t = x_s$
- $\tau(\neg\phi) \doteq \neg\tau(\phi)$
- $\tau(\phi \lor \psi) \doteq \tau(\phi) \lor \tau(\psi)$
- $\tau(\textbf{if } \phi \textbf{ then } \psi \textbf{ else } \chi \textbf{ fi}) \doteq \textbf{if } \tau(\phi) \textbf{ then } \tau(\psi) \textbf{ else } \tau(\chi) \textbf{ fi}$
- $\tau(\exists x : \phi) \doteq \exists x : \tau(\phi)$
- $\tau(\forall \bar{x} : P(\bar{t}) \leftarrow \phi) \doteq \forall \bar{x}, \bar{x}_{\bar{t}} : P(\bar{x}_{\bar{t}}) \leftarrow G(\bar{t}, \bar{x}_{\bar{t}}) \land \tau(\phi)$
- $\tau(\forall \bar{x} : f(\bar{t}) = k \leftarrow \phi) \doteq \forall \bar{x}, \bar{x}_{\bar{t}}, x_k : \gamma_f(\bar{x}_{\bar{t}}, x_k) \leftarrow G(\bar{t}, \bar{x}_{\bar{t}}) \land G(k, x_k) \land \tau(\phi)$
- $\tau(\Phi) \doteq \{\tau(r) \mid r \in \Phi\} \cup$
  $\{\forall \bar{x} : \delta_f(\bar{x}) \leftarrow \exists y : \gamma_f(\bar{x}, y) \mid f \text{ is a defined function symbol of } \Phi\}$

As before, $\bar{x}_{\bar{t}} \doteq x_{t_1}, \dots, x_{t_n}$ and x_k are designated variables that are solely used to refer to the terms $\bar{t} \doteq t_1, \dots, t_n$ and k respectively, and $G(\bar{t}, \bar{x}_{\bar{t}})$ is shorthand notation for $\bigwedge_{i=1}^{n} G(t_i, x_{t_i})$.

The translation function τ eliminates all occurrences of partial function symbols in a formula and replaces them by existential statements involving the auxiliary function G from Definition 24. The translation $\tau(\Phi)$ of a definition Φ contains a rule $\forall \bar{x}, \bar{x}_{\bar{t}}, x_k : \gamma_f(\bar{x}_{\bar{t}}, x_k) \leftarrow G(\bar{t}, \bar{x}_{\bar{t}}) \land G(k, x_k) \land \tau(\phi)$ defining the graph predicate γ_f for every rule $\forall \bar{x} : f(\bar{t}) = k \leftarrow \phi$ in Φ defining a (partial) function f/n, as well as a rule $\forall \bar{x} : \delta_f(\bar{x}) \leftarrow \exists y : \gamma_f(\bar{x}, y)$ defining the corresponding domain predicate δ_f as the projection of γ_f onto its first n components.

Remark 3 Strictly following the rules of previous definitions can lead to rather verbose translations. However, one can typically (manually) simplify translations in a semantics-preserving way. For instance, if a tuple $\bar{t}$ of terms consists only of variables and total constant symbols (e.g., 0), then $\tau(P(\bar{t})) \doteq \exists \bar{x}_{\bar{t}} : G(\bar{t}, \bar{x}_{\bar{t}}) \land P(\bar{x}_{\bar{t}})$ can be simplified to $P(\bar{t})$. Abusing notation, we will also write $\tau(\phi)$ to refer to a simplification of the translation of ϕ.

Finally, to come to the notion of a model of an FO(RD) definition, we translate FO(PF) structures to FO structures. This is done by simply removing all partial function symbols from the vocabulary, working instead with the associated graph and domain predicate symbols.

Definition 26 The *translation* $\tau(\mathfrak{A})$ of an FO(PF) structure $\mathfrak{A}$ is the FO structure obtained by removing all partial function symbols from its vocabulary.[24] We say that an FO(PF) structure $\mathfrak{A}$ *satisfies* or is a *model* of an FO(RD) formula ϕ, and write $\mathfrak{A} \models \phi$, if $\tau(\mathfrak{A}) \models \tau(\phi)$. A Φ-*context* $\mathcal{O}$ is a structure whose vocabulary consists of all parameters of Φ. We say that an FO(RD) definition Φ is *total in* $\mathcal{O}$ if Φ has a model that extends $\mathcal{O}$. We say that Φ is *total* if it is total in every Φ-context.

Example 25 Let Φ_{minus} be the definition from Example 16. Strictly applying the rules of Definitions 24 and 25, the first definitional rule of Φ_{minus} translates to

$$\begin{aligned} &\forall n, x_n, x_0 : \gamma_{minus}(x_n, x_0, x_n) \leftarrow G(n, x_n) \wedge G(0, x_0) \\ \doteq\ &\forall n, x_n, x_0 : \gamma_{minus}(x_n, x_0, x_n) \leftarrow x_n = n \wedge x_0 = 0. \end{aligned}$$

However, this can be simplified to

$$\forall n : \gamma_{minus}(n, 0, n).$$

Similarly, the translation of the second rule simplifies to

$$\begin{aligned} &\forall n, m, x_1 : \gamma_{minus}(n, s(m), x_1) \leftarrow G(p(minus(n, m)), x_1) \\ \doteq\ &\forall n, m, x_1 : \gamma_{minus}(n, s(m), x_1) \leftarrow \exists x_2 : G(minus(n, m), x_2) \wedge \gamma_p(x_2, x_1) \\ \doteq\ &\forall n, m, x_1 : \gamma_{minus}(n, s(m), x_1) \leftarrow \exists x_2 : \gamma_{minus}(n, m, x_2) \wedge \gamma_p(x_2, x_1). \end{aligned}$$

Here we replaced the variables $x_{p(minus(n,m))}$ and $x_{minus(n,m)}$ by x_1 and x_2 respectively, and we used that s is a total function symbol. The translation $\tau(\Phi_{minus})$ of the definition Φ_{minus} becomes

$$\left\{ \begin{array}{l} \forall n : \gamma_{minus}(n, 0, n) \\ \forall n, m, x_1 : \gamma_{minus}(n, s(m), x_1) \leftarrow \exists x_2 : \gamma_{minus}(n, m, x_2) \wedge \gamma_p(x_2, x_1) \\ \forall n, m : \delta_{minus}(n, m) \leftarrow \exists y : \gamma_{minus}(n, m, y) \end{array} \right\}.$$

Consider the Φ_{minus}-context $\mathcal{O}$ with universe $\mathbb{N}$ and the standard interpretation for 0, s and p (and γ_p). The well-founded model of Φ_{minus} in $\mathcal{O}$ interprets γ_{minus} as the subset of $\mathbb{N}^3$ consisting of all triples $(n, m, n - m)$ with $n \geq m$, and δ_{minus} as the subset of $\mathbb{N}^2$ consisting of all pairs (n, m) with $n \geq m$. Thus, every extension of $\mathcal{O}$ that interprets γ_{minus} and δ_{minus} as these respective sets is a model of $\tau(\Phi_{minus})$. Consequently, every FO(PF) structure that extends such a model by interpreting *minus* as the function described in Example 16 is a model of Φ_{minus}.

Example 26 Consider the definition Φ_d from Example 21. After (quite) some simplifications, the translation $\tau(\Phi_d)$ of Φ_d becomes

$$\left\{ \begin{array}{l} \forall n : \gamma_d(n, n, 0) \\ \forall n, m, p, x_1 : \gamma_d(n, m, s(x_1)) \leftarrow \gamma_d(n, p, x_1) \wedge E(p, m) \wedge \neg\exists q : E(q, m) \wedge \\ \hfill \exists x_2 : \gamma_d(n, q, x_2) \wedge x_2 < x_1 \\ \forall n, m : \delta_d(n, m) \leftarrow \exists y : \gamma_d(n, m, y) \end{array} \right\}.$$

Here we replaced the variables $x_{d(n,p)}$ and $x_{d(n,q)}$ by x_1 and x_2 respectively, and we used that 0 is a total constant symbol and s a total function symbol. Furthermore, we left out the atoms $\delta_d(n, p)$ and $\delta_d(n, q)$ in the body of the second rule, since they are already implied by the atoms $\gamma_d(n, p, x_1)$ and $\gamma_d(n, q, x_2)$ respectively.

Note that $\tau(\Phi_d)$ is a non-monotone and moreover a non-stratified definition, as the defined predicate γ_d occurs in the scope of a negation in the second rule. As mentioned in Section 7.1, this definition cannot be represented in several other logics for inductive definitions.

[24]Note that all information about partial function interpretations $f^{\mathfrak{A}}$ in $\mathfrak{A}$ is still present in $\tau(\mathfrak{A})$ through the corresponding graph predicate interpretations $\gamma_f^{\mathfrak{A}}$.

Example 27 The translation of definition Φ_{second} from Example 24 can be simplified to

$$\left\{ \begin{array}{l} \gamma_{second}(\mathit{fish}) \leftarrow \gamma_{\mathit{first}}(\mathit{pasta}) \land \neg \mathit{Friday} \\ \forall x : \gamma_{second}(x) \leftarrow \gamma_{\mathit{first}}(x) \land \mathit{Friday} \\ \delta_{second} \leftarrow \exists x : \gamma_{second}(x) \end{array} \right\}.$$

The definition can be encoded into ASP as follows:

```
g_second(fish) :- g_first(pasta), not Friday.
g_second(X) :- g_first(X), Friday.
d_second :- g_second(X).
```

This ASP encoding does not include the functionality constraints for the graph predicates, which are implicit in the FO(PF) structure. To ensure the correct behavior when using an ASP solver, we add the following two constraints. These enforce that each of the graph relations contains at most one object.

```
:- g_first(X), g_first(Y), X != Y.
:- g_second(X), g_second(Y), X != Y.
```

In FO(RD), non-totality can additionally arise from the overdefinition of a (partial) function. Partial functions have the restriction that at most one function value can be assigned to any tuple of arguments. Hence, when defining a partial function, one needs to take care that this condition is met. For example, the FO(RD) definition $\{\, \forall x, y : f(x) = y \,\}$ is non-total in any structure whose universe contains more than one element. Its translation $\{\, \forall x, y : \gamma_f(x, y) \,\}$ to FO(ID) is total in any structure, and its well-founded model interprets γ_f as the Cartesian product $\mathcal{U} \times \mathcal{U}$ of the universe $\mathcal{U}$ with itself. However, as soon as $\mathcal{U}$ contains more than one element, γ_f no longer corresponds to the graph of a (partial) function. In general, a model $\mathfrak{B}$ of $\tau(\Phi)$ extends to a model of Φ if and only if $\mathfrak{B}$ interprets the defined graph predicates $\gamma_f/n+1$ as graphs, i.e., such that for every $\bar{x} \in \mathcal{U}^n$, there is at most one $y \in \mathcal{U}$ such that $(\bar{x}, y) \in \gamma_f^{\mathfrak{B}}$.

7.3 Functional programming

Recursive definitions of functions commonly appear in functional programming. Chapter 6 of the book *Denotational Semantics* by Schmidt [43] provides a formal account of recursive definitions in functional programming, which we will refer to as FP.

In FP, recursive function definitions are specified by means of *lambda expressions*:

$$f = \lambda \bar{x}.\, t \tag{5}$$

where f is a function symbol, $\bar{x}$ a tuple of variables, and t a term that may contain f and $\bar{x}$. The syntax of the term t is defined as before, but with the addition of the so-called *choice function* $\_ \to \_ \,\Box\, \_$:

- if ϕ is a formula not containing any quantifiers or f, and t and s are terms, then $\phi \to t \,\Box\, s$ is a term.

The term $\phi \to t \,\square\, s$ is equal to t if ϕ evaluates to true, and is equal to s if ϕ evaluates to false. The reason why we do not allow f to occur in ϕ is to prevent paradoxical examples such as the following:

$$f = \lambda x.\ f(x) = 0 \to 1 \,\square\, 0$$

This lambda expression says that $f(x)$ is equal to 1 if it is equal to 0 and that it is equal 0 if it is not equal to 0, which does not specify a well-defined function.

Example 28 The half function from Example 14 can be defined in FP by the following lambda expression:

$$\mathit{half} = \lambda n.\ n = 0 \to 0 \,\square\, s(\mathit{half}(p(p(n)))) \tag{6}$$

where s denotes the successor function and p the predecessor function. The expression says that half sends 0 to itself, and every non-zero number n to $s(\mathit{half}(p(p(n))))$. Note that $s(\mathit{half}(p(p(n))))$ is not always defined. Here we follow the convention that a function applied to an undefined term is undefined, which is explained in more formal detail in Section 7.3.1.

In FP, functions are first-class citizens, which can be passed as arguments to other functions. This is not possible in FO(RD). On the other hand, FO(RD) allows the use of FO bodies in its function definitions, which is not the case for FP. For instance, the definition of *spouse* from Example 22 is not representable in FP, as it requires the introduction of a variable y that is not a function of the argument variable x. Note that the definition from Example 22 possibly overdefines *spouse*, in case the predicate *Married* would contain pairs $(a, b), (a, c)$ with $b \neq c$. Such overdefinedness of functions is not possible in the framework of FP. Similarly, the definitions from Examples 21 and 23 are not representable in FP.

A lambda expression (5) can be translated to an FO(RD) definition as follows:

$$\{\ \forall \bar{x} : f(\bar{x}) = t \leftarrow \Delta(t)\ \} \tag{7}$$

Here we extend Definition 10 of Δ with the following rule:

- $\Delta(\phi \to t \,\square\, s) \doteq \Delta(\bar{r}) \overrightarrow{\wedge} (\textbf{if } \phi \textbf{ then } \Delta(t) \textbf{ else } \Delta(s) \textbf{ fi})$, where $\bar{r}$ are the unguarded terms in ϕ.

In the remainder of this section, we will provide a semantics for (a certain class of) the lambda expressions in the FP framework introduced in Section 7.3, based on [43, Chapter 6]. Furthermore, we will provide a simplified semantics for the corresponding definitions of graph predicates in FO(ID), and we will show a strong correspondence result between both semantics.

7.3.1 Semantics of FP

In FP, functions are defined on *domains*, which can be *primitive* (such as the set of natural numbers $\mathbb{N}$ or Boolean values $\mathbb{B}$) or *compound*. Compound domains are built from domains A, B as either *products* $A \times B$, *disjoint unions* (or *sums*) $A + B$, *function spaces* $A \to B$, or *liftings* $A_\perp$. The *lifting* $A_\perp$ of a domain A is obtained by adding a *bottom element* $\perp$ to A, representing undefinedness or non-termination. A partial function from A to B can be seen as a total function from A to $B_\perp$, which takes the value $\perp$ precisely on its undefined elements.

Henceforth, we fix a primitive domain D as well as a lambda expression

$$f = \lambda \bar{x}.\, t \tag{8}$$

defining a function in $D^k \to D$. A formal semantics for the lambda expression (8) is given by a least fixpoint construction.

Let $\mathfrak{B}$ be an FO(PF) structure with universe $D_\perp = D \cup \{\perp\}$ that interprets all non-logical symbols in the lambda expression (8). Assume that $\mathfrak{B}$ interprets any n-ary function symbol g as a function in $D_\perp^n \to D_\perp$ that maps any tuple $\bar{a} \in D_\perp^n$ containing a bottom element $\perp$ to $\perp$.[25] Furthermore, we follow the convention that $(\phi \to s_1 \square s_2)^{\mathfrak{B}} = \perp$ if ϕ contains a term that evaluates to $\perp$. Let $\mathfrak{B}^f$ be the set of expansions of $\mathfrak{B}$ with interpretations for f as functions $D_\perp^k \to D_\perp$ that map any tuple $\bar{a} \in D_\perp^k$ containing a bottom element $\perp$ to $\perp$. We define the *definedness order* $\sqsubseteq$ on $D_\perp$ by

$$x \sqsubseteq y \text{ iff } x = \perp \text{ or } x = y$$

This partial order extends pointwise to a partial order on the function space $D_\perp^k \to D_\perp$, and moreover to a partial order on $\mathfrak{B}^f$ as follows:

$$\mathcal{J} \sqsubseteq \mathcal{J}' \text{ iff } f^{\mathcal{J}} \sqsubseteq f^{\mathcal{J}'} \text{ iff } f^{\mathcal{J}}(\bar{a}) \sqsubseteq f^{\mathcal{J}'}(\bar{a}) \text{ for all } \bar{a} \in D_\perp^k$$

We furthermore define an operator $\mathcal{G}$ on $\mathfrak{B}^f$ by

$$f^{\mathcal{G}(\mathcal{J})}(\bar{a}) = t^{\mathcal{J}[\bar{x}:\bar{a}]} \text{ for all } \bar{a} \in D^k.$$

(Note that this fully determines the structure $\mathcal{G}(\mathcal{J})$ as $f^{\mathcal{G}(\mathcal{J})}(\bar{a}) = \perp$ for any $\bar{a} \in D_\perp^k \setminus D^k$ and the interpretation of all other non-logical symbols is fixed by $\mathfrak{B}$, by definition of $\mathfrak{B}^f$.) It is straightforward to see that $\mathcal{G}$ is monotone, meaning that $\mathcal{G}(\mathcal{J}) \sqsubseteq \mathcal{G}(\mathcal{J}')$ for $\mathcal{J} \sqsubseteq \mathcal{J}'$. We define a sequence $\langle \mathcal{J}_i \rangle_{i \le \omega}$ of structures in $\mathfrak{B}^f$ by letting

$$\begin{aligned} \mathcal{J}_0 &= \perp_{\mathfrak{B}^f}, \\ \mathcal{J}_{i+1} &= \mathcal{G}(\mathcal{J}_i), \\ \mathcal{J}_\omega &= \bigsqcup_{i<\omega} \mathcal{J}_i. \end{aligned}$$

Here $\perp_{\mathfrak{B}^f}$ denotes the least element of $\mathfrak{B}^f$, which interprets f as the function sending every element of $D_\perp^k$ to $\perp$, and $\bigsqcup_{i<\omega} \mathcal{J}_i$ denotes the least upper bound of all $\mathcal{J}_i$ with $i < \omega$. This least upper bound is well-defined since $\mathcal{J}_i \sqsubseteq \mathcal{J}_j$ for $i < j$, which follows from the fact that $\mathcal{G}$ is monotone. We will use f_i as shorthand notation for $f^{\mathcal{J}_i}$. The function f_ω is the declarative meaning we assign to the lambda expression (8).

[25] Here we refer to $(D_\perp)^n$ simply as $D_\perp^n$.

Example 29 Reconsider the lambda expression (6) from Example 28. For this expression, $f = \mathit{half}$, $D = \mathbb{N}$, $k = 1$ and $\mathfrak{B}$ is the structure with universe $\mathbb{N}_\perp$ and the standard interpretation for 0, s and p. The functions half_i are as follows:

$$\mathit{half}_0(a) = \perp,$$

$$\mathit{half}_1(a) = \Big(n = 0 \rightarrow 0 \,\square\, s(\mathit{half}(p(p(n))))\Big)^{\mathcal{J}_0[n:a]} = \begin{cases} 0 & \text{if } a = 0, \\ \perp & \text{otherwise,} \end{cases}$$

$$\mathit{half}_2(a) = \Big(n = 0 \rightarrow 0 \,\square\, s(\mathit{half}(p(p(n))))\Big)^{\mathcal{J}_1[n:a]} = \begin{cases} 0 & \text{if } a = 0, \\ 1 & \text{if } a = 2, \\ \perp & \text{otherwise,} \end{cases}$$

$$\vdots$$

$$\mathit{half}_{i+1}(a) = \Big(n = 0 \rightarrow 0 \,\square\, s(\mathit{half}(p(p(n))))\Big)^{\mathcal{J}_i[n:a]} = \begin{cases} \frac{a}{2} & \text{if } a \in 2\mathbb{N} \text{ and } s \leq 2i, \\ \perp & \text{otherwise,} \end{cases}$$

$$\vdots$$

$$\mathit{half}_\omega(a) = \bigsqcup_{i<\omega} \mathit{half}_i(a) = \begin{cases} \frac{a}{2} & \text{if } a \in 2\mathbb{N}, \\ \perp & \text{otherwise,} \end{cases}$$

for all $a \in \mathbb{N}_\perp$. Note that half_ω is the expected function corresponding to (6).

7.3.2 Simplified semantics for graph predicate definitions

The lambda expression (8) in FP is translated to the following definition in FO(RD):

$$\left\{ \forall \bar{x} : f(\bar{x}) = t \leftarrow \Delta(t) \right\}. \tag{9}$$

In turn, this is translated to the following definition in FO(ID):

$$\left\{ \begin{array}{l} \forall \bar{x}, x_t : \gamma_f(\bar{x}, x_t) \leftarrow G(t, x_t) \\ \forall \bar{x} : \delta_f(\bar{x}) \leftarrow \exists y : \gamma_f(\bar{x}, y) \end{array} \right\} \tag{10}$$

Here we extend Definition 24 of G with the following rule:

- $G(\phi \rightarrow t \,\square\, s, y) \doteq \tau(\phi) \rightarrow G(t, y) \,\square\, G(s, y)$

We left out the atom $\Delta(t)$, since it is implied by $G(t, x_t)$. We will show that the models of the lambda expression in FP correspond to the models of its translation to FO(RD) (7).

Since $G(t, x_t)$ contains no negation, (10) is a *positive* definition, which entails that the intricate well-founded semantics simplifies to standard monotone induction for this definition [36]. This standard monotone induction consists of a repeated application of a monotone operator to structures interpreting γ_f and δ_f until the first infinite ordinal ω, similar as for FP. For simplicity, we will focus on the graph predicate γ_f, as the domain predicate δ_f is determined in terms of γ_f.

Let $\mathfrak{A}$ be the FO structure with universe D that corresponds to structure $\mathfrak{B}$ from Section 7.3.1 in the following way: for every partial function symbol g/l interpreted by $\mathfrak{B}$, $\mathfrak{A}$ interprets the corresponding graph predicate symbol $\gamma_g/l{+}1$ such that

$$(\bar{a}, b) \in \gamma_g^{\mathfrak{A}} \text{ iff } g^{\mathfrak{B}}(\bar{a}) = b \tag{11}$$

for all $\bar{a} \in D^l, b \in D$.

Let $\mathfrak{A}^{\gamma_f}$ denote the set of all expansions of $\mathfrak{A}$ with an interpretation for the $(k+1)$-ary predicate symbol γ_f. We define a partial order $\sqsubseteq$ on $\mathfrak{A}^{\gamma_f}$ by letting $\mathcal{I} \sqsubseteq \mathcal{I}'$ iff $\gamma_f^{\mathcal{I}} \subseteq \gamma_f^{\mathcal{I}'}$. Furthermore, we define an operator $\mathcal{G}$ on $\mathfrak{A}^{\gamma_f}$ by

$$\gamma_f^{\mathcal{G}(\mathcal{I})} = \{(\bar{a}, b) \in D^{k+1} \mid \mathcal{I}[\bar{x} : \bar{a}, x_t : b] \models G(t, x_t)\}.$$

(Note that this fully determines the structure $\mathcal{G}(\mathcal{I})$ as the interpretation of the other non-logical symbols is fixed by $\mathfrak{A}$.) Since $G(t, x_t)$ is a positive formula, it is straightforward to see that $\mathcal{G}$ is monotone, i.e., $\mathcal{G}(\mathcal{I}) \sqsubseteq \mathcal{G}(\mathcal{I}')$ if $\mathcal{I} \sqsubseteq \mathcal{I}'$. We define a sequence $\langle \mathcal{I}_i \rangle_{i \leq \omega}$ of structures in $\mathfrak{A}^{\gamma_f}$ by letting

$$\begin{aligned} \mathcal{I}_0 &= \bot_{\mathfrak{A}^{\gamma_f}}, \\ \mathcal{I}_{i+1} &= \mathcal{G}(\mathcal{I}_i), \\ \mathcal{I}_\omega &= \bigsqcup_{i<\omega} \mathcal{I}_i. \end{aligned}$$

Here $\bot_{\mathfrak{A}^{\gamma_f}}$ denotes the least element of $\mathfrak{A}^{\gamma_f}$, which interprets γ_f as the empty set, and $\bigsqcup_{i<\omega} \mathcal{I}_i$ denotes the least upper bound of all $\mathcal{I}_i$ with $i < \omega$. It is well-defined since $\mathcal{I}_i \sqsubseteq \mathcal{I}_j$ for $i < j$, which follows from the fact that $\mathcal{G}$ is monotone. We will use γ_f^i as shorthand notation for $\gamma_f^{\mathcal{I}_i}$. The interpretation of γ_f by the well-founded model of (10) is γ_f^ω.

7.3.3 Correspondence result

We will now prove that the interpretations of f in both frameworks correspond in a strong sense.

Definition 27 Let $\mathcal{I} \in \mathfrak{A}^{\gamma_f}$ and $\mathcal{J} \in \mathfrak{B}^f$. We write $\mathcal{I} \simeq \mathcal{J}$ if for all $\bar{a} \in D^k, b \in D$: $(\bar{a}, b) \in \gamma_f^{\mathcal{I}}$ iff $f^{\mathcal{J}}(\bar{a}) = b$.

Lemma 1 *Let $\mathcal{I} \in \mathfrak{A}^{\gamma_f}$ and $\mathcal{J} \in \mathfrak{B}^f$. If $\mathcal{I} \simeq \mathcal{J}$, then $\mathcal{G}(\mathcal{I}) \simeq \mathcal{F}(\mathcal{J})$.*

Proof Assume that $\mathcal{I} \simeq \mathcal{J}$, and take $\bar{a} \in D^k, b \in D$ arbitrarily. We need to show that $(\bar{a}, b) \in \gamma_f^{\mathcal{G}(\mathcal{I})}$ iff $f^{\mathcal{F}(\mathcal{J})}(\bar{a}) = b$. By the definitions of $\mathcal{G}$ and $\mathcal{F}$, this comes down to

$$\mathcal{I}[\bar{x} : \bar{a}, x_t : b] \models G(t, x_t) \text{ iff } t^{\mathcal{J}[\bar{x}:\bar{a}]} = b. \tag{12}$$

We prove this equivalence by structural induction on the term t.

Case 1: Suppose that t is a variable y that is interpreted by both $\mathcal{I}[\bar{x} : \bar{a}, x_t : b]$ and $\mathcal{J}[\bar{x} : \bar{a}]$. Then y must be one of the variables among $\bar{x}$, say x_i. The right side of (12) simplifies to $a_i = b$. The left side of (12) simplifies to $\mathcal{I}[\bar{x} : \bar{a}, x_t : b] \models x_i = x_t$, which is equivalent to $a_i = b$.

Case 2: Suppose that t is of the form $g(s_1, \dots, s_n)$, and that the equivalence (12) holds for $s_1, \dots, s_n$ (induction hypothesis). Suppose that $\mathcal{I}[\bar{x} : \bar{a}, x_t : b] \models G(t, x_t)$. Then $\mathcal{I}[\bar{x} : \bar{a}, x_t : b] \models \exists \bar{x}_{\bar{s}} : G(\bar{s}, \bar{x}_{\bar{s}}) \wedge \gamma_g(\bar{x}_{\bar{s}}, x_t)$. This means there exist $\bar{d} \in D^n$ such that $\mathcal{I}[\bar{x} : \bar{a}, x_t : b, \bar{x}_{\bar{s}} : \bar{d}] \models G(\bar{s}, \bar{x}_{\bar{s}}) \wedge \gamma_g(\bar{x}_{\bar{s}}, x_t)$. In particular, $\mathcal{I}[\bar{x} : \bar{a}, x_{s_i} : d_i] \models G(s_i, x_{s_i})$ for all i, which implies that $s_i^{\mathcal{J}[\bar{x}:\bar{a}]} = d_i$ by the induction hypothesis. Furthermore, the fact that $\mathcal{I}[x_t : b, \bar{x}_{\bar{s}} : \bar{d}] \models \gamma_g(\bar{x}_{\bar{s}}, x_t)$ implies that $g^{\mathcal{J}}(\bar{d}) = b$. This follows from the fact that $\mathcal{I} \simeq \mathcal{J}$ in case $g \doteq f$, and from (11) otherwise. Together, these facts imply that $t^{\mathcal{J}[\bar{x}:\bar{a}]} = g^{\mathcal{J}}(\bar{s}^{\mathcal{J}[\bar{x}:\bar{a}]}) = b$.

Suppose conversely that $t^{\mathcal{J}[\bar{x}:\bar{a}]} = b$. Then every $s_i^{\mathcal{I}[\bar{x}:\bar{a}]} = d_i$ for some $d_i \in D$. (Note that $d_i \neq \bot$, for otherwise $\bot = g^{\mathcal{J}}(d_1, \dots, d_n) = b \in D$; a contradiction.) It follows by the induction hypothesis that $\mathcal{I}[\bar{x} : \bar{a}, x_{s_i} : d_i] \models G(s_i, x_{s_i})$ for every i. Furthermore, $g^{\mathcal{J}}(\bar{d}) = b$ implies $(\bar{d}, b) \in \gamma_g^{\mathcal{I}}$. This follows from the fact that $\mathcal{I} \simeq \mathcal{J}$ in case $h = f$ and from (11) otherwise. Combining these findings, we obtain $\mathcal{I}[\bar{x} : \bar{a}, x_t : b] \models \exists \bar{x}_{\bar{s}} : G(\bar{s}, \bar{x}_{\bar{s}}) \wedge \gamma_g(\bar{x}_{\bar{s}}, x_t)$, and hence $\mathcal{I}[\bar{x} : \bar{a}, x_t : b] \models G(t, x_t)$.

Case 3: Suppose that t is of the form $\phi \rightarrow s_1$ ▯ s_2, and that the equivalence (12) holds for s_1 and s_2 (induction hypothesis). Then the equivalence for t follows from the induction hypothesis after observing that $\mathcal{J}[\bar{x} : \bar{a}] \models \phi$ iff $\mathcal{I}[\bar{x} : \bar{a}, x_t : b] \models \tau(\phi)$ (recall function τ from Definition 25 eliminates all partial function symbols by replacing them with their graphs). This finishes the proof. □

Theorem 7 $\mathcal{I}_\omega \simeq \mathcal{J}_\omega$.

Proof This follows from the more general statement that $\mathcal{I}_i \simeq \mathcal{J}_i$ for all $i \leq \omega$, which is proved by induction on i. For $i = 0$, this is trivial, as $\gamma_f^0 = \emptyset$ and $f_0(\bar{a}) = \bot$ for all $\bar{a} \in D^k$. The induction step follows from Lemma 7. Finally, for $i = \omega$, it suffices to show that $\simeq$ is preserved under taking least upper bounds, which is straightforward. Indeed, for arbitrary $\bar{a} \in D^k$ and $b \in D$, we have that $f_\omega(\bar{a}) = b$ iff $f_j(\bar{a}) = b$ for some $j < \omega$ iff $(\bar{a}, b) \in \gamma_f^j$ for some $j < \omega$ iff $(\bar{a}, b) \in \gamma_f^\omega$. □

8 Conclusion

In this paper, we developed a prudence-based logic for partial functions with language constructs designed to eliminate ambiguities introduced by non-denoting terms. These constructs, ranging from conditional guards to asymmetric connectives and annotations, serve to ensure the well-definedness of expressions containing partial functions. Furthermore, they are designed to make well-definedness a decidable syntactical property. Hence, guarding conditions of our logic deliver the principle of well-definedness: although the underlying logic is three-valued, guarded formulas are "error-free" in the sense that their truth value is well-defined in all structures, as established in Theorem 2. Additionally, Theorem 6 demonstrates that our approach allows for more general graph unnesting of function terms compared to classical logic with partial functions. In the final part of the paper, we extended the language of our logic with definitions of partial functions. The semantics of these definitions were reduced to the semantics of FO(ID), via a translation that replaces all partial functions by the corresponding graph predicates. Additionally, we investigated the relation between our logic and functional programming.

The approach to partial functions introduced in this paper is tailored towards the formal logic languages used for Knowledge Representation (KR). The language FO(PF) offers clear insights into the constructs involved in guarding partial functions. Additionally, we explored classical logic and rule-based definitions that form the theoretical foundations of most symbolic KR systems. As a result, the concepts presented in this paper can be applied to many existing systems with minimal adaptations.

This paper focuses on the foundations of systematically treating partial functions in logic-based KR languages. KR languages are naturally paired with solvers that handle the reasoning process, and the efficiency of these solvers is crucial. The fact that well-definedness in our language is decidable (in polynomial time, as shown in Theorem 3) contributes to solver efficiency. However, the impact of introducing a domain predicate for each function on the solver's performance is not explored in this paper.

Acknowledgements. We would like to sincerely thank Maurice Bruynooghe for his valuable feedback and suggestions throughout the entire process of creating this paper and Pierre Carbonnelle for many useful discussions. We also thank Linde Vanbesien for pointing out an error in Example 21.

Funding. This work was partially supported by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen", Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G070521N), Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G0B2221N), and KUL research project C14/19/082.

Declarations

Data availability. No datasets were generated or analysed during the current study.

Conflict of interest. The authors declare that they have no conflict of interest.

Appendix A Semantics of FO(ID)

Most parts of this appendix appear literally in [37].

Following [22], we define the semantics of FO(ID) based on the well-founded semantics from logic programming [41].

In FO(ID), structures $\mathfrak{A}$ are defined as usual, consisting of a set $\mathcal{U}$ called the *universe* of $\mathfrak{A}$, and a mapping from non-logical symbols σ to *interpretations* $\sigma^{\mathfrak{A}}$ in $\mathcal{U}$. Constant symbols are interpreted as elements of $\mathcal{U}$, n-ary predicate symbols as subsets of $\mathcal{U}^n$, and n-ary function symbols as functions $\mathcal{U}^n \to \mathcal{U}$. The set of interpreted symbols of $\mathfrak{A}$ is called the *vocabulary* of $\mathfrak{A}$, and we speak of a Σ*-structure* to refer to a structure with vocabulary Σ. Given a structure $\mathfrak{A}$ with universe $\mathcal{U}$, a tuple of non-logical symbols $\bar{\sigma}$, and a tuple of corresponding values $\bar{\alpha}$ in $\mathcal{U}$, we denote by $\mathfrak{A}[\bar{\sigma} : \bar{\alpha}]$ the structure obtained from $\mathfrak{A}$ by interpreting $\bar{\sigma}$ by $\bar{\alpha}$. If Σ is a subset of the vocabulary of $\mathfrak{A}$, then $\mathfrak{A}|_\Sigma$ denotes the structure obtained from $\mathfrak{A}$ by restricting its interpretation mapping to Σ.

To define the semantics of definitions, Denecker and Vennekens [22] use *three-valued structures*, in which there are three different *truth values*: $\mathbf{f}$, $\mathbf{u}$ and $\mathbf{t}$, standing for 'false', 'unknown' and 'true' respectively. We consider two orders on the set of truth values: the *truth order* $\leq$ and the *precision order* $\leq_p$, defined by $\mathbf{f} \leq \mathbf{u} \leq \mathbf{t}$, and $\mathbf{u} \leq_p \mathbf{f}$, $\mathbf{u} \leq_p \mathbf{t}$ respectively. A *three-valued structure* $\mathcal{I}$ is defined similarly as a regular (two-valued) structure, except that n-ary predicate symbols are interpreted as functions from $\mathcal{U}^n$ to $\{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$, where $\mathcal{U}$ is the universe of $\mathcal{I}$. We extend the truth and precision order pointwise to functions with codomain $\{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$, and furthermore to three-valued structures by letting $\mathcal{I} \leq \mathcal{J}$ if $\mathcal{I}$ and $\mathcal{J}$ have the same universe and vocabulary, the same interpretation for non-predicate symbols, and $P^{\mathcal{I}} \leq P^{\mathcal{J}}$ for every predicate symbol P, and similarly for $\leq_p$.

Note that a two-valued structure can be seen as a special case of a three-valued structure, by identifying a subset S of $\mathcal{U}^n$ with its characteristic function, sending every element of S to $\mathbf{t}$ and every non-element to $\mathbf{f}$. In this way, two-valued structures correspond to maximally precise three-valued structures. In the sequel, we will frequently abuse this correspondence, for instance by writing $P^{\mathfrak{A}}(\bar{a}) = \mathbf{t}$ instead of $\bar{a} \in P^{\mathfrak{A}}$ for two-valued structures $\mathfrak{A}$.

Henceforth, following [22], we fix a definition Φ. We denote the set of defined predicates of Φ by $\mathrm{def}(\Phi)$, the set of parameters of Φ by $\mathrm{pars}(\Phi)$, and their union by $\mathrm{sym}(\Phi)$. (Hence, $\mathrm{sym}(\Phi)$ is the set of all predicate symbols that occur in Φ.) We also fix a $\mathrm{pars}(\Phi)$-structure $\mathcal{O}$, which we call a Φ-*context*, both with universe $\mathcal{U}$. Furthermore, we fix a *(three-valued) truth function*, sending each pair $(\phi, \mathcal{I})$ of an FO formula ϕ and a three-valued structure $\mathcal{I}$ to a truth value $\phi^{\mathcal{I}} \in \{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$. We let this truth function be a generic one, imposing only two conditions:

- $\mathcal{I} \leq_p \mathcal{J}$ implies $\phi^{\mathcal{I}} \leq_p \phi^{\mathcal{J}}$ ($\leq_p$-monotonicity).
- If $\mathcal{I}$ is two-valued, then $\phi^{\mathcal{I}}$ takes the standard truth value of ϕ in $\mathcal{I}$.

Given a predicate symbol P, we denote by n_P the arity of P. Given a defined predicate P of Φ, we write

$$\varphi_P(\bar{y}) \doteq \bigvee \exists \bar{x} : \bar{y} = \bar{t} \wedge \varphi, \tag{A1}$$

where the disjunction ranges over all rules $\forall \bar{x} : P(\bar{t}) \leftarrow \varphi$ in Φ defining P, and $\bar{y}$ is an n_P-tuple of object symbols not occurring in any of these rules. Given a tuple of non-logical symbols $\bar{\sigma}$ (possibly) occurring in a formula ϕ, together with a tuple of corresponding values $\bar{\alpha}$ in $\mathcal{U}$, we write $\phi(\bar{\alpha}/\bar{\sigma})^{\mathcal{I}}$, or shortly $\phi(\bar{\alpha})^{\mathcal{I}}$, to refer to $\phi^{\mathcal{I}[\bar{\sigma}:\bar{\alpha}]}$. A *universe atom* is a pair $(P, \bar{a})$, where P is a predicate symbol and $\bar{a} \in \mathcal{U}^{n_P}$. With an abuse of notation, we will often write $P(\bar{a})$ instead of $(P, \bar{a})$. Given a set of predicate symbols Π, we denote by $At^{\Pi}_{\mathcal{U}}$ the set of universe atoms $(P, \bar{a})$ for which $P \in \Pi$.

Definition 28 Let $\mathcal{I}$ be a three-valued $\mathrm{sym}(\Phi)$-structure with universe $\mathcal{U}$. A three-valued $\mathrm{sym}(\Phi)$-structure $\mathcal{I}'$ is said to be a Φ-*refinement* of $\mathcal{I}$ if there exists a non-empty set $U \subseteq At^{\mathrm{def}(\Phi)}_{\mathcal{U}}$ such that $P^{\mathcal{I}}(\bar{a}) = \mathbf{u}$ for all $P(\bar{a}) \in U$, and either

- $\mathcal{I}' = \mathcal{I}[U : \mathbf{t}]$ and for all $P(\bar{a}) \in U$, $\phi_P^{\mathcal{I}}(\bar{a}) = \mathbf{t}$, or
- $\mathcal{I}' = \mathcal{I}[U : \mathbf{f}]$ and for all $P(\bar{a}) \in U$, $\phi_P^{\mathcal{I}'}(\bar{a}) = \mathbf{f}$.

Here $\mathcal{I}[U : \mathbf{t}]$ refers to the structure identical to $\mathcal{I}$, except that its interpretation of P maps $\bar{a}$ to $\mathbf{t}$ for all $P(\bar{a}) \in U$, and similarly for $\mathcal{I}[U : \mathbf{f}]$.

Note the asymmetry between deriving truth and falsity of defined atoms: the truth of defined atoms can only be derived if the corresponding bodies are true in $\mathcal{I}$, while the falsity of defined atoms may be derived if this implies the falsity of the corresponding bodies in $\mathcal{I}'$. The second kind of refinement is based on the idea of unfounded sets, as introduced in [41].

Definition 29 A *well-founded induction* of Φ in $\mathcal{O}$ is a sequence $\langle \mathcal{I}_i \rangle_{0 \leq i \leq \beta}$ of three-valued sym(Φ)-structures extending $\mathcal{O}$ such that $P^{\mathcal{I}_0} = \mathbf{u}$ for all $P \in \text{def}(\Phi)$,[26] $\mathcal{I}_{i+1}$ is a Φ-refinement of $\mathcal{I}_i$ for every ordinal $0 \leq i < \beta$, and $\mathcal{I}_\lambda$ is the $\leq_p$-limit of $\langle \mathcal{I}_i \rangle_{i<\lambda}$ for every limit ordinal $0 < \lambda \leq \beta$. A well-founded induction $\langle \mathcal{I}_i \rangle_{0 \leq i \leq \beta}$ is called *terminal* if its *limit* $\mathcal{I}_\beta$ has no Φ-refinement.

Remark 4 Note that well-founded inductions are $\leq_p$-increasing, which guarantees that the $\leq_p$-limits are well-defined. Since the truth function is $\leq_p$-monotone, well-founded inductions $\langle \mathcal{I}_i \rangle_{0 \leq i \leq \beta}$ have the property that once $\phi^{\mathcal{I}_i} \in \{\mathbf{f}, \mathbf{t}\}$ for some i, then $\phi^{\mathcal{I}_i} = \phi^{\mathcal{I}_j}$ for all $j \geq i$.

Theorem 8 (Denecker and Vennekens, 2007 [42]) *Any definition has a terminal well-founded induction in any context. Furthermore, all terminal well-founded inductions of a given definition in a given context have the same limit.*

Definition 30 The *well-founded model* of Φ in $\mathcal{O}$ is the limit of a well-founded induction of Φ in $\mathcal{O}$. If this well-founded model is two-valued, we say that Φ is *total* in $\mathcal{O}$. If Φ is total in any Φ-context, we say that Φ is *total*.

Example 30 Consider the following definition of the even number predicate $E/1$ in terms of the constant 0 and successor function s:

$$\Phi_E = \left\{ \begin{array}{l} E(0) \\ \forall n : E(s(n)) \leftarrow \neg E(n) \end{array} \right\}$$

Let $\mathcal{O}$ be the pars(Φ_E)-structure with as universe the set of natural numbers $\mathbb{N}$, and with the standard interpretation for 0 and s. Then

$$\begin{aligned}
&E^{\mathcal{I}_0}(n) = \mathbf{u} \text{ for } n \in \mathbb{N} \\
&E^{\mathcal{I}_1}(0) = \mathbf{t}, \quad E^{\mathcal{I}_1}(n) = \mathbf{u} \text{ for } n \in \mathbb{N} \setminus \{0\} \\
&E^{\mathcal{I}_2}(0) = \mathbf{t},\ E^{\mathcal{I}_2}(1) = \mathbf{f},\ E^{\mathcal{I}_2}(n) = \mathbf{u} \text{ for } n \in \mathbb{N} \setminus \{0, 1\} \\
&\qquad\vdots \\
&E^{\mathcal{I}_\omega}(n) = \mathbf{t} \text{ for } n \in 2\mathbb{N}, \quad E^{\mathcal{I}_\omega}(n) = \mathbf{f} \text{ for } n \in 2\mathbb{N} + 1
\end{aligned}$$

[26] Here $P^{\mathcal{I}_0} = \mathbf{u}$ means that $P^{\mathcal{I}_0}(\bar{a}) = \mathbf{u}$ for all $\bar{a} \in \mathcal{U}^{n_P}$.

specifies a terminal well-founded induction $\langle \mathcal{I}_i \rangle_{0 \leq i \leq \omega}$ of Φ_E in $\mathcal{O}$. Here ω denotes the first infinite ordinal. Since the well-founded model $\mathcal{I}_\omega$ is two-valued, Φ_E is total in $\mathcal{O}$.

However, Φ_E is not total in general. In an adapted context $\mathcal{O}'$ in which $s^{\mathcal{O}'}(1) = 1$ for instance, the only well-founded induction stagnates at step 1. Indeed, in this case $\mathcal{I}_1$ has no refinements, as assigning **f** to $E(1)$ makes $\phi_{E(1)} \doteq 1 = 0 \vee \exists x(1 = s(x) \wedge \neg E(x))$ true. Thus, the well-founded interpretation of Φ_E in $\mathcal{O}'$ is three-valued, and therefore Φ_E is non-total in $\mathcal{O}'$.

The satisfaction relation for two-valued structures in FO(ID) is defined as for FO, but with the addition of the following clause:

- $\mathfrak{A} \models \Phi$ if $\mathfrak{A}|_{\mathrm{sym}(\Phi)}$ is the well-founded model of Φ in $\mathfrak{A}|_{\mathrm{pars}(\Phi)}$.

A *tautology* in FO(ID) is a formula that is satisfied by every two-valued structure interpreting its non-logical symbols.

References

[1] Karttunen, L.: Presupposition and linguistic context. Theoretical Linguistics **1**(1-3), 181–194 (1974) https://doi.org/10.1515/thli.1974.1.1-3.181

[2] Russell, B.: On denoting. Mind **14**(56), 479–493 (1905)

[3] Frege, G.: On sense and reference. na (1892). https://doi.org/10.1515/9783111546216-003

[4] Strawson, P.F.: On referring. Mind **59**(235), 320–344 (1950)

[5] Suppes, P.: Introduction to Logic. Dover books on mathematics. Dover Publications, Mineola, (1999)

[6] Kleene, S.C.: Introduction to Metamathematics. P. Noordhoff N.V., Groningen, (1952)

[7] Schock, R.: Logics Without Existence Assumptions. Stockholm, Sweden: Stockholm, Almqvist & Wiksell, Stockholm, (1968)

[8] Fitting, M.: Kleene's three valued logics and their children. Fundamenta informaticae **20**(1, 2, 3), 113–131 (1994) https://doi.org/10.3233/FI-1994-201234

[9] Gavilanes-Franco, A., Lucio-Carrasco, F.: A first order logic for partial functions. Theoretical Computer Science **74**(1), 37–69 (1990) https://doi.org/10.1016/0304-3975(90)90005-3

[10] de Nivelle, H.: Classical logic with partial functions. In: International Joint Conference on Automated Reasoning, pp. 203–217 (2010). https://doi.org/10.1007/978-3-642-14203-1_18 . Springer

[11] McCarthy, J.: A basis for a mathematical theory of computation. In: Computer Programming and Formal Systems. Studies in Logic and the Foundations of Mathematics, vol. 26, pp. 33–70. Elsevier, Amsterdam, (1959)

[12] Berezin, S., Barrett, C., Shikanian, I., Chechik, M., Gurfinkel, A., Dill, D.L.: A practical approach to partial functions in CVC lite. Electronic Notes in Theoretical Computer Science **125**(3), 13–23 (2005) https://doi.org/10.1016/J.ENTCS.2004.06.064

[13] Hoang, T.S.: An introduction to the Event-B modelling method. Industrial Deployment of System Engineering Methods, 211–236 (2013)

[14] Cristiá, M., Rossi, G., Frydman, C.S.: Adding partial functions to Constraint Logic Programming with sets. Theory and Practice of Logic Programming **15**, 651–665 (2015) https://doi.org/10.1017/S1471068415000290

[15] Frisch, A.M., Stuckey, P.J.: The proper treatment of undefinedness in constraint languages. In: International Conference on Principles and Practice of Constraint Programming, pp. 367–382 (2009). https://doi.org/10.1007/978-3-642-04244-7_30 . Springer

[16] De Cat, B., Bogaerts, B., Bruynooghe, M., Janssens, G., Denecker, M.: Predicate logic as a modeling language: the IDP system. In: Declarative Logic Programming: Theory, Systems, and Applications, pp. 279–323. Association for Computing Machinery and Morgan & Claypool, New York, (2018). https://doi.org/10.1145/3191315.3191321

[17] Cabalar, P.: Partial functions and equality in answer set programming. In: Logic Programming: 24th International Conference, ICLP 2008 Udine, Italy, December 9-13 2008 Proceedings 24, pp. 392–406 (2008). https://doi.org/10.1007/978-3-540-89982-2_36 . Springer

[18] Balduccini, M.: A "conservative" approach to extending answer set programming with non-Herbrand functions. Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz, 24–39 (2012) https://doi.org/10.1007/978-3-642-30743-0_3

[19] Balduccini, M.: ASP with non-Herbrand partial functions: a language and system for practical use. Theory and Practice of Logic Programming **13**(4-5), 547–561 (2013) https://doi.org/10.1017/S1471068413000343

[20] Farmer, W.M.: A partial functions version of Church's simple theory of types. The Journal of Symbolic Logic **55**(3), 1269–1291 (1990) https://doi.org/10.2307/2274487

[21] Markovic, D., Bruynooghe, M., Denecker, M.: Towards systematic treatment of partial functions in knowledge representation. In: Gaggl, S., Martinez, M.V.,

Ortiz, M. (eds.) Logics in Artificial Intelligence, pp. 756–770. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-43619-2_51

[22] Denecker, M., Vennekens, J.: The well-founded semantics is the principle of inductive definition, revisited. In: Chitta, B., De Giacomo, G. (eds.) Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning. AAAI Press, Palo Alto, (2014)

[23] Northrop, E.P., Silver, D.S.: Riddles in Mathematics: A Book of Paradoxes. Courier Corporation, Mineola, (2014)

[24] Cobreros, P., Égré, P., Ripley, D., Rooij, R.v.: Foreword: Three-valued logics and their applications. Journal of Applied Non-Classical Logics **24**(1-2), 1–11 (2014) https://doi.org/10.1080/11663081.2014.909631

[25] Gabbay, D.M., Woods, J.: The Many Valued and Nonmonotonic Turn in Logic. Elsevier, Amsterdam, (2007)

[26] Heyting, A.: Die formalen regeln der intuitionistischen logik. Sitzungsbericht PreuBische Akademie der Wissenschaften Berlin, physikalisch-mathematische Klasse II, 42–56 (1930)

[27] Pearce, D.: A new logical characterisation of stable models and answer sets. In: Dix, J., Pereira, L.M., Przymusinski, T.C. (eds.) Non-Monotonic Extensions of Logic Programming, pp. 57–70. Springer, Berlin, Heidelberg (1997). https://doi.org/10.1007/BFB0023801

[28] Gödel, K.: Zum intuitionistischen aussagenkalkül. Anzeiger der Akademie der Wissenschaften in Wien **69**, 65–66 (1932)

[29] Scott, D.: Identity and existence in intuitionistic logic. In: Applications of Sheaves: Proceedings of the Research Symposium on Applications of Sheaf Theory to Logic, Algebra, and Analysis, Durham, July 9–21, 1977, pp. 660–696 (2006). Springer

[30] Church, A.: A formulation of the simple theory of types. The journal of symbolic logic **5**(2), 56–68 (1940) https://doi.org/10.2307/2266170

[31] Bove, A., Dybjer, P.: Dependent types at work. In: International LerNet ALFA Summer School on Language Engineering and Rigorous Software Development, pp. 57–99. Springer, Berlin, Heidelberg, (2008). https://doi.org/10.1007/978-3-642-03153-3_2

[32] Wittocx, J., Mariën, M., Denecker, M.: Grounding FO and FO(ID) with bounds. Journal of Artificial Intelligence Research **38**, 223–269 (2010) https://doi.org/10.1613/JAIR.2980

[33] Lonsing, F., Egly, U.: Evaluating QBF solvers: Quantifier alternations matter. In: Principles and Practice of Constraint Programming: 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings 24, pp. 276–294 (2018). https://doi.org/10.1007/978-3-319-98334-9_19 . Springer

[34] Rabe, M.N., Seshia, S.A.: Incremental determinization. In: Theory and Applications of Satisfiability Testing–SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings 19, pp. 375–392 (2016). https://doi.org/10.1007/978-3-319-40970-2_23 . Springer

[35] Denecker, M.: Extending Classical Logic with Inductive Definitions (2000). https://doi.org/10.1007/3-540-44957-4_47

[36] Denecker, M., Ternovska, E.: A logic of non-monotone inductive definitions. ACM transactions on computational logic (TOCL) **9**(2), 1–52 (2008) https://doi.org/10.1145/1342991.1342998

[37] Van den Eede, R., Van Biervliet, R., Denecker, M.: A sequent calculus for generalized inductive definitions. In: Dodaro, C., Gupta, G., Martinez, M.V. (eds.) Logic Programming and Nonmonotonic Reasoning, pp. 30–42. Springer, Cham (2025)

[38] Martin-Löf, P.: Hauptsatz for the intuitionistic theory of iterated inductive definitions. In: Studies in Logic and the Foundations of Mathematics vol. 63, pp. 179–216. Elsevier, Amsterdam, (1971). https://doi.org/10.1016/S0049-237X(08)70847-4

[39] McDowell, R., Miller, D.: Cut-elimination for a logic with definitions and induction. Theoretical Computer Science **232**(1), 91–119 (2000) https://doi.org/10.1016/S0304-3975(99)00171-1

[40] Momigliano, A., Tiu, A.: Induction and co-induction in sequent calculus. In: Berardi, S., Coppo, M., Damiani, F. (eds.) Types for Proofs and Programs, pp. 293–308. Springer, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24849-1_19

[41] Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM **38**(3), 619–649 (1991) https://doi.org/10.1145/116825.116838

[42] Denecker, M., Vennekens, J.: Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In: Baral, C., Brewka, G., Schlipf, J. (eds.) Logic Programming and Nonmonotonic Reasoning, pp. 84–96. Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72200-7_9

[43] Schmidt, D.A.: Denotational Semantics: a methodology for language development. Wm. C. Brown Publishers (1988)