symmetry

MDPI

*Article*

# A Novel Categorical Approach to Semantics of Relational First-Order Logic

**Wolfgang Schreiner [1] and William Steingartner [2,*] and Valerie Novitzká [2]**

[1] Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Altenbergerstraße 69, A-4040 Linz, Austria; wolfgang.schreiner@risc.jku.at
[2] Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia; valerie.novitzka@tuke.sk
* Correspondence: william.steingartner@tuke.sk

Received: 30 August 2020; Accepted: 18 September 2020; Published: 23 September 2020

check for updates

**Abstract:** We present a categorical formalization of a variant of first-order logic. Unlike other texts on this topic, the goal of this paper is to give a very transparent and self-contained account without requiring more background than basic logic and set theory. Our focus is to show how the semantics of first-order formulas can be derived from their usual deduction rules. For understanding the core ideas, it is not necessary to investigate the internal term structure of atomic formulas, thus we abstract atomic formulas to (syntactically opaque) relations; in this sense, our variant of first-order logic is "relational". While the derived semantics is based on categorical principles (even the duality that arises from a symmetry between two ways of looking at something where there is no reason to choose one over the other), it is nevertheless "constructive" in that it describes explicit computations of the truth values of formulas. We demonstrate this by modeling the categorical semantics in the RISCAL (RISC Algorithm Language) system which allows us to validate the core propositions by automatically checking them in finite models.

**Keywords:** category; functor; RISCAL; relation; relational first-order logic; semantics

## 1. Introduction

Most introductions to first-order logic first define the syntax of formulas, then formalize their meaning in the form established in the 1930s by Tarski [1] (essentially what we call today in programming language theory a "denotational semantics" [2]), then introduce a deduction calculus, and finally show the soundness and completeness of this calculus concerning the semantics: if a formula can be derived in the calculus, it is true according to the semantics, and vice versa. These relationships between truth and derivability have to be established because there is no self-evident link between the semantics of a formula and the deduction rules associated with it. Historically, deduction came first; the soundness of a deduction calculus was established by showing that it could not lead to apparent inconsistencies, i.e. that both a formula and its negation could not be derived in a deduction system. It was Tarski who first gave meaning to formulas that was independent of deduction.

However, as it did since the 1940s to many other mathematical areas, category theory [3–7], the general theory of mathematical structures, can bring and provide an alternative light also to first-order logic. It does so by considering logical notions as special instances of "universal" constructions, where a value of interest is determined

- first, by depicting the core property that the value shall satisfy; and
- second, by giving a criterion how to choose a canonical value from all values that satisfy the property.

It was eventually recognized that by such universal constructions the semantics of the connectives of propositional logic could be determined directly from their associated introduction and elimination rules. However, it took until the late 1960s until Lawvere gained the fundamental insight that this idea could be also applied to the quantifiers of first-order logic [8], thus establishing a direct relationship between its semantics and its proof calculus.

However, this insight has not yet obtained a foothold in basic texts on logic and its basic education. The main reason may be that the corresponding material is found mostly in texts on category theory and its applications where it is dispersed among examples of the application of categorical notions without a clear central presentation. Furthermore, the general treatment of first-order logic with terms and variables requires a complex mathematical apparatus [9] which is much beyond the scope of basic introductions. Reasonably compact introductions can be found, e.g., in Section 2.1.10 of [10], in [11], in Section 1.6 of [12] (however, in the context of type theory rather than classical first-order logic), in Section 9.5 of [3] (the treatment of quantifiers only), and in Section 7.1.12 of [7] (again only the treatment of quantifiers).

The goal of this paper is to give a compact introduction to a categorical version of first-order logic that is fully self-contained, only introduces the categorical notions relevant for the stated purpose, and presents them from the point of view of the intended application. For this purpose, it elaborates a simple but completely formalized syntactic and semantic framework of first-order logic that represents the background of the discussion, without gaps and inconsistencies. As a deliberate decision, this framework does not address the syntax and semantics of terms but abstracts atomic formulas to opaque relations; this allows for focusing the discussion on the essentials. However, to describe a reasonably close relative of first-order logic, this framework is (in contrast to other presentations) not based on relations of fixed arity, i.e., with a fixed number of variables; instead, we consider relations of infinite arity, i.e., with infinitely many variables. However, only finitely many variables may influence the truth value of the relation, which represents the effect that a classical atomic formula can only reference a finite number of variables. The overall result is a slick and elegant presentation. Because the duality has many manifestations in logic and it is agr eed by all hands that a duality is like a "giant symmetry"—a symmetry between theories, we focus on this concept in our approach. For the implementation, we use the RISCAL—the *RISC Algorithm Language* [13], which is a specification language with an associated software system for describing mathematical algorithms, formally specifying their behavior based on mathematical theories, and validating the correctness of algorithms, specifications, and theories by the execution/evaluation of their formal semantics. The term "algorithm language" indicates that RISCAL is intended to model, rather than low-level code, algorithms (as can be found in textbooks on discrete mathematics) in a high-level language, and specifying the behavior of these algorithms by formal contacts. RISCAL has been developed to validate the correctness of mathematical theories, specifications, and programs, by checking instances of these artifacts on finite domains; applications of RISCAL are for instance in discrete mathematics, number theory, and computer algebra. Software based on formal logic plays an ever-increasing role in areas where a mathematically precise understanding of a subject domain and sound rules for reasoning about the properties of this domain are essential. A prime example is the formal modeling, specification, and verification of computer programs and computing systems, but there are many other applications in areas such as knowledge-based systems, computer mathematics, or the semantic web [14]. Furthermore, the intent of all our projects (namely LogTechEdu, SemTech [15], and others listed in Funding section) is to further advance education in computer science and related topics. In academical courses for computer science and mathematics, by utilizing the power of modern software based on formal logic and semantics, students shall engage with the material they encounter by actively producing the problem solutions rather than just passively taking them from the lecturer.

The remainder of this paper is structured as follows: in Section 2, we define a term-free variant of first-order logic and give it a semantics in the usual style based on set-theoretic notions. In Section 3, we introduce those categorical notions that are necessary for understanding the following elaboration and discuss their relationships. The core of this paper is Section 4 where we elaborate the categorical formulation of the semantics of our variant of first-order logic. In Section 5, we demonstrate that these semantics are constructive by modeling it in the RISCAL system [13], which allows us to automatically check the core propositions in particular finite models. Section 6 concludes our presentation and gives an outlook on our future work.

## 2. A Relational First-Order Logic

In this section, we introduce a simplified variant of first-order logic that abstracts from the syntactic structure of atomic formulas and thus copes without the concept of terms, constants, function symbols, predicate symbols, and all of the associated semantic apparatus. Towards this goal, atomic formulas are replaced by relations over assignments (maps of variables to values) that are constrained to only depend on a finite number of variables; we will call such relations "predicates". Consequently, the semantics of every non-atomic formula is also a relation (i.e., a predicate, as mentioned).

We begin with some standard notions. First, we specify variables and the values that variables hold.

**Axiom 1** (Variables and Values)**.** *Let Var denote an arbitrary infinite and enumerable set; we call the elements of this set variables. Furthermore, let Val denote an arbitrary non-empty set; we call the elements of these set values.*

Next, we define assignments.

**Definition 1** (Assignments)**.** *We define $Ass := Var \to Val$ as the set of all mappings of variables to values (a function space); we call the elements of this set assignments. Thus, for every assignment $a \in Ass$ and every variable $x \in Var$, we have $a(x) \in Val$.*

We note that assignment is similar to a concept of state in theory of formal semantics of programming languages (see, e.g., [16]) where the state is a function from variables to values: to each variable, the state associates its current value.

**Definition 2** (Updates)**.** *Let $a \in Ass$ be an assignment, $x \in Var$ a variable, and $v \in Val$ a value. We define the update assignment $a[x \mapsto v] \in Ass$ as follows:*

$$a[x \mapsto v](y) := \begin{cases} v, & \text{if } x = y; \\ a[x], & \text{otherwise.} \end{cases}$$

*Consequently, $a[x \mapsto v]$ is identical to a except that it maps variable x to value v.*

Based on this, we can formulate the following updating properties.

**Proposition 1** (Update Properties)**.** *Let $a \in Ass$ be an assignment, $x, y \in Var$ variables, and $v, v_1, v_2 \in Val$ values. Then, we have the following properties:*

$$a[x \mapsto a(x)] = a,$$
$$a[x \mapsto v_1][x \mapsto v_2] = a[x \mapsto v_2],$$
$$x \neq y \Rightarrow a[x \mapsto v_1][y \mapsto v_2] = a[y \mapsto v_2][x \mapsto v_1].$$

**Proof.** Directly from the definitions. □

The properties of assignments listed above (and only these) will be of importance in the subsequent proofs.

Now, we turn to the fundamental semantic notions.

**Definition 3** (Relations). *We define $Rel := \mathcal{P}(Ass)$ as the set of all sets of assignments; we call the elements of this set "relations". Consequently, a relation is a set of assignments.*

**Definition 4** (Variable Independence). *We state that relation $R \in Rel$ is independent of variable $x \in Var$, written as $R \perp\!\!\!\perp x$, if and only if the following holds:*

$$\forall a \in Ass, v_1, v_2 \in Val.\ a[x \mapsto v_1] \in R \Leftrightarrow a[x \mapsto v_2] \in R.$$

*Consequently, if $R \perp\!\!\!\perp x$, the value of $x$ in any assignment $a$ does not influence whether $a$ is in $R$. We say that $R$ depends on $x$ if $R \perp\!\!\!\perp x$ does not hold.*

We transfer the central syntactic property of atomic formulas (they can only refer to finitely many variables) to its semantic counterpart.

**Definition 5** (Predicates). *A relation $R \in Rel$ is a* predicate*, if it only depends on finitely many variables. We denote by Pred the set of all predicates and by $Pred_x := \{P \in Pred \mid P \perp\!\!\!\perp x\}$ the subset of all predicates that are independent of $x$.*

Now, we are ready to introduce the central entities of our paper. First, we give a definition of the abstract syntax of formulas.

**Definition 6** (Abstract Syntax of Formulas). *We define For as that smallest set of abstract syntax trees in which every element $F \in For$ is generated by an application of a rule of the following context-free grammar (where $P \in Pred$ denotes an arbitrary predicate and $x \in Var$ denotes an arbitrary variable):*

$$\begin{aligned} F ::= {} & P \mid \top \mid \bot \\ & \mid \neg F \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid F_1 \rightarrow F_2 \mid F_1 \leftrightarrow F_2 \\ & \mid \forall x.\ F \mid \exists x.\ F \end{aligned}$$

*We call the elements of this set formulas.*

In this definition, the role of a classic atomic predicate $p(t_1, \ldots, t_n)$ with argument terms $t_1, \ldots, t_k$ in which $n$ variables $x_1, \ldots, x_n$ occur freely is abstracted to a predicate $P$ that depends on variables $x_1, \ldots, x_n$.

Now, we establish the relationship between the syntax and semantics of formulas.

**Definition 7** (Semantics of Formulas). *Let $F \in For$ be a formula. We define the relation $[\![ F ]\!] \in Rel$, called the* semantics *of $F$, by induction on the structure of $F$:*

$$\begin{aligned} [\![ P ]\!] &:= \{a \in Ass \mid a \in P\} \\ [\![ \top ]\!] &:= \{a \in Ass \mid true\} \\ [\![ \bot ]\!] &:= \{a \in Ass \mid false\} \\ [\![ \neg F ]\!] &:= \{a \in Ass \mid a \notin [\![ F ]\!]\} \\ [\![ F_1 \wedge F_2 ]\!] &:= \{a \in Ass \mid a \in [\![ F_1 ]\!] \text{ and } a \in [\![ F_2 ]\!]\} \\ [\![ F_1 \vee F_2 ]\!] &:= \{a \in Ass \mid a \in [\![ F_1 ]\!] \text{ or } a \in [\![ F_2 ]\!]\} \\ [\![ F_1 \rightarrow F_2 ]\!] &:= \{a \in Asss \mid a \notin [\![ F_1 ]\!] \text{ or } a \in [\![ F_2 ]\!]\} \\ [\![ F_1 \leftrightarrow F_2 ]\!] &:= \{a \in Ass \mid \big(a \in [\![ F_1 ]\!] \text{ and } a \in [\![ F_2 ]\!]\big) \text{ or } \big(a \notin [\![ F_1 ]\!] \text{ and } a \notin [\![ F_2 ]\!]\big)\} \\ [\![ \forall x.\ F ]\!] &:= \{a \in Ass \mid a[x \mapsto v] \in [\![ F ]\!], \text{ for all } v \in Val\} \\ [\![ \exists x.\ F ]\!] &:= \{a \in Ass \mid a[x \mapsto v] \in [\![ F ]\!], \text{ for some } v \in Val\} \end{aligned}$$

The above definition is well-defined in that every formula denotes a relation. To show that formulas indeed denote predicates, some more work is required.

**Proposition 2** (Quantified Formulas and Variable Independence)**.** *For every variable $x \in Var$ and formula $F \in For$, we have $[\![ \forall x.\ F ]\!] \perp\!\!\!\perp x$ and $[\![ \exists x.\ F ]\!] \perp\!\!\!\perp x$, i.e., the semantics of quantified formulas do not depend on $x$.*

**Proof.** We prove this proposition by reductio ad absurdum.

First, assume that $[\![ \forall x.\ F ]\!]$ depends on $x$. Then, we have some assignment $a \in [\![ \forall x.\ F ]\!]$ and some values $v_1, v_2 \in Val$ such that $a[x \mapsto v_1] \in [\![ \forall x.\ F ]\!]$ and $a[x \mapsto v_2] \notin [\![ \forall x.\ F ]\!]$. From $a[x \mapsto v_2] \notin [\![ \forall x.\ F ]\!]$ we have some $v \in Val$ with $a[x \mapsto v_2][x \mapsto v] \notin [\![ F ]\!]$ and thus $a[x \mapsto v] \notin [\![ F ]\!]$. However, $a[x \mapsto v_1] \in [\![ \forall x.\ F ]\!]$ implies $a[x \mapsto v_1][x \mapsto v] \in [\![ F ]\!]$ and thus $a[x \mapsto v] \in [\![ F ]\!]$, which represents a contradiction.

Now, assume that $[\![ \exists x.\ F ]\!]$ depends on $x$. Then, we have some assignment $a \in [\![ \exists x.\ F ]\!]$ and some values $v_1, v_2 \in Val$ such that $a[x \mapsto v_1] \in [\![ \exists x.\ F ]\!]$ and $a[x \mapsto v_2] \notin [\![ \exists x.\ F ]\!]$. From $a[x \mapsto v_1] \in [\![ \exists x.\ F ]\!]$, we have some $v \in Val$ with $a[x \mapsto v_1][x \mapsto v] \in [\![ F ]\!]$ and thus $a[x \mapsto v] \in [\![ F ]\!]$. However, $a[x \mapsto v_2] \notin [\![ \exists x.\ F ]\!]$ implies $a[x \mapsto v_2][x \mapsto v] \notin [\![ F ]\!]$ and thus $a[x \mapsto v] \notin [\![ F ]\!]$, which represents a contradiction. □

**Proposition 3** (Formula Semantics and Predicates)**.** *For every formula $F \in For$, we have $[\![ F ]\!] \in Pred$, i.e., the semantics of $F$ is a predicate.*

**Proof.** The proof proceeds by induction over the structure of $F$.

- If $F = P$, we have $[\![ F ]\!] = \{a \in Ass \mid a \in P\} = P \in Pred$.
- If $F \in \{\top, \bot\}$, there are no $x, a, v_1, v_2$ such that $a[x \mapsto v_1] \in [\![ F ]\!]$ and $a[x \mapsto v_2] \notin [\![ F ]\!]$ because, for $F = \top$, the second condition must be false and for $F = \bot$ the first one; thus, $F$ does not depend on any variable.
- If $F = \neg F_1$, by the induction hypothesis, we may assume that $[\![ F_1 ]\!]$ depends only on the variables in some finite variable set $X$. From the definition of $[\![ F ]\!]$, it is then easy to show that $[\![ \neg F_1 ]\!]$ also depends only on the variables in $X$.
- If $F \in \{F_1 \wedge F_2, F_1 \vee F_2, F_1 \rightarrow F_2, F_1 \leftrightarrow F_2\}$, we may assume by the induction hypothesis that $[\![ F_1 ]\!]$ depends only on the variables in some finite set $X_1$ while $P_2$ only depends on the variables in some finite set $X_2$. From the definition of $[\![ F ]\!]$, it is then easy to show that $[\![ F ]\!]$ depends only on the variables in the finite set $X_1 \cup X_2$.
- If $F \in \{\forall x.\ F_1, \exists x.\ F_1\}$, we may assume by the induction hypothesis that $[\![ F_1 ]\!]$ only depends on the variables in some finite variable set $X$. We are now going to show that $[\![ F ]\!]$ only depends on the variables in the finite set $X \backslash \{x\}$. Actually, we assume that this is not the case and show a contradiction. From this assumption and Proposition 2, we have a variable $y \neq x \wedge y \notin X$ on which $F$ depends; thus, we have an assignment $a$ and values $v_1, v_2$ such that $a[y \mapsto v_1] \in [\![ F ]\!]$ and $a[y \mapsto v_2] \notin [\![ F ]\!]$.

  If $F = \forall x.\ F_1$, from $a[y \mapsto v_2] \notin [\![ F ]\!]$, we have a $v \in Val$ with $a[y \mapsto v_2][x \mapsto v] \notin [\![ F_1 ]\!]$ and thus (since $y \neq x$) $a[x \mapsto v][y \mapsto v_2] \notin [\![ F_1 ]\!]$. From $a[y \mapsto v_1] \in [\![ F ]\!]$, we know $a[y \mapsto v_1][x \mapsto v] \in [\![ F_1 ]\!]$ and thus $a[x \mapsto v][y \mapsto v_1] \in [\![ F_1 ]\!]$. Thus, $[\![ F_1 ]\!]$ depends on a variable $y \notin X$ which contradicts the induction assumption.

  If $F = \exists x.\ F_1$, from $a[y \mapsto v_1] \in [\![ F ]\!]$, we have a $v \in Val$ with $a[y \mapsto v_1][x \mapsto v] \in [\![ F_1 ]\!]$ and thus (since $y \neq x$) $a[x \mapsto v][y \mapsto v_1] \in [\![ F_1 ]\!]$. From $a[y \mapsto v_2] \notin [\![ F ]\!]$ we know $a[y \mapsto v_2][x \mapsto v] \notin [\![ F_1 ]\!]$ and thus $a[x \mapsto v][y \mapsto v_2] \notin [\![ F_1 ]\!]$. Thus, $[\![ F_1 ]\!]$ depends on a variable $y \notin X$ which contradicts the induction assumption.

This completes our proof. □

In the following, we transfer the classical model-theoretic notions to our framework.

**Definition 8** (Satisfaction). *Let $a \in Ass$ be an assignment and $F \in For$ be a formula. We define $a \models F$ (read: a satisfies F) as follows:*

$$a \models F :\Leftrightarrow a \in [\![ F ]\!].$$

**Definition 9** (Validity). *Let $F \in For$ be a formula. We define $\models F$ (read: F is valid) as follows:*

$$\models F :\Leftrightarrow \forall a \in Ass.\ a \models F.$$

**Definition 10** (Logical Consequence). *Let $F, G \in For$ be formulas. We define $F \models G$ (read: G is a logical consequence of F) as follows:*

$$F \models G :\Leftrightarrow \forall a \in Ass.\ a \models F \Rightarrow a \models G.$$

**Definition 11** (Logical Equivalence). *Let $F, G \in For$ be formulas. We define $F \equiv G$ (read: F and G are logically equivalent) as follows:*

$$F \equiv G :\Leftrightarrow \forall a \in Ass.\ a \models F \Leftrightarrow a \models G.$$

**Proposition 4** (Logical Consequence and Logical Equivalence). *Let $F, G \in For$ be formulas. Then, we have the following equivalences:*

- $(F \models G) \Leftrightarrow (\models F \rightarrow G)$
- $(F \models G) \Leftrightarrow ([\![ F ]\!] \subseteq [\![ G ]\!])$
- $(F \equiv G) \Leftrightarrow (\models F \leftrightarrow G)$
- $(F \equiv G) \Leftrightarrow ([\![ F ]\!] = [\![ G ]\!])$

**Proof.** Directly from the definitions. □

Thus, a logical consequence on the meta-level coincides with an implication on the formula level and with the subset relation on the semantic level. Furthermore, logical equivalence on the meta-level coincides with equivalence on the formula level and with the equality relation on the semantic level.

In the following, we establish a set-theoretic interpretation of the logical operations of our formula language.

**Definition 12** (Complement). *We define the* complement $\overline{R} \in Rel$ *of relation* $R \in Rel$ *as the relation* $\overline{R} := Ass \backslash R$*. Consequently, an assignment is in* $\overline{R}$ *if and only if it is not in* $R$.

**Proposition 5** (Propositional Semantics as Set Operations). *Let $F, F_1, F_2 \in For$ be formulas. We then have the following equalities:*

$$\begin{aligned}
[\![ P ]\!] &= P \\
[\![ \top ]\!] &= Ass \\
[\![ \bot ]\!] &= \emptyset \\
[\![ \neg F ]\!] &= \overline{[\![ F ]\!]} \\
[\![ F_1 \wedge F_2 ]\!] &= [\![ F_1 ]\!] \cap [\![ F_2 ]\!] \\
[\![ F_1 \vee F_2 ]\!] &= [\![ F_1 ]\!] \cup [\![ F_2 ]\!] \\
[\![ F_1 \rightarrow F_2 ]\!] &= \overline{[\![ F_1 ]\!]} \cup [\![ F_2 ]\!] \\
[\![ F_1 \leftrightarrow F_2 ]\!] &= \big([\![ F_1 ]\!] \cap [\![ F_2 ]\!]\big) \cup \big(\overline{[\![ F_1 ]\!]} \cap \overline{[\![ F_2 ]\!]}\big)
\end{aligned}$$

**Proof.** Directly from the definition of the semantics. □

While the above results are quite intuitive, a corresponding set-theoretic interpretation of quantified formulas is not. In the following, we only state the plain result without indication of how it can be intuitively understood; we will delegate this explanation to Section 4, where the categorical framework will provide us with adequate insight.

**Proposition 6** (Quantifier Semantics as Set Operations)**.** *Let $F \in For$ be a formula. We then have the following equalities:*

$$\llbracket \forall x.\, F \rrbracket = \bigcup \{P \in Pred \mid P \perp\!\!\!\perp x \land P \subseteq \llbracket F \rrbracket\},$$
$$\llbracket \exists x.\, F \rrbracket = \bigcap \{P \in Pred \mid P \perp\!\!\!\perp x \land \llbracket F \rrbracket \subseteq P\}.$$

*In other words, $\llbracket \forall x.\, F \rrbracket$ is the weakest predicate $P$ ("weakest" in the sense of the largest set) that is independent from $x$ and that satisfies the property $P \subseteq \llbracket F \rrbracket$ while $\llbracket \exists x.\, F \rrbracket$ is the strongest predicate $P$ ("strongest" in the sense of the smallest set) that is independent of $x$ and that satisfies the property $\llbracket F \rrbracket \subseteq P$.*

**Proof.** The proof is in two stages. First, we take an arbitrary assignment $a \in Ass$ and show

$$a \in \llbracket \forall x.\, F \rrbracket \Leftrightarrow \big(\exists P \in Pred.\, P \perp\!\!\!\perp x \land P \subseteq \llbracket F \rrbracket \land a \in P\big)$$

$\Rightarrow$: We assume $a \in \llbracket \forall x.\, F \rrbracket$ and prove for $P := \llbracket \forall x.\, F \rrbracket$

$$P \in Pred \tag{1}$$
$$P \perp\!\!\!\perp x \tag{2}$$
$$P \subseteq \llbracket F \rrbracket \tag{3}$$
$$a \in P \tag{4}$$

From Proposition 3, we have (1). From Proposition 2, we have (2). From $a \in \llbracket \forall x.\, F \rrbracket$, we have (4). To show (3), we take arbitrary assignment $a_0 \in P$ and show $a_0 \in \llbracket F \rrbracket$. From $a_0 \in P$, we know $a_0[x \mapsto v] \in \llbracket F \rrbracket$ for $v := a_0(x)$. Since $a_0[x \mapsto a_0(x)] = a_0$, we thus know $a_0 \in \llbracket F \rrbracket$.

$\Leftarrow$: We assume for some $P \in Pred$

$$P \perp\!\!\!\perp x \tag{5}$$
$$P \subseteq \llbracket F \rrbracket \tag{6}$$
$$a \in P \tag{7}$$

and prove $a \in \llbracket \forall x.\, F \rrbracket$. For this, we take arbitrary $v \in Val$ and prove $a[x \mapsto v] \in \llbracket F \rrbracket$. From (6), it suffices to show $a[x \mapsto v] \in P$. From (5), we know

$$\forall v_1, v_2 \in Val.\, a[x \mapsto v_1] \in P \Leftrightarrow a[x \mapsto v_2] \in P \tag{8}$$

From (7) and $a = a[x \mapsto a(x)]$, we know $a[x \mapsto v_0] \in P$ for $v_0 := a(x)$. Thus, with (8), we know $a[x \mapsto v] \in P$.

Now, we take arbitrary $a \in Ass$ and show

$$a \in \llbracket \exists x.\, F \rrbracket \Leftrightarrow \big(\forall P \in Pred.\, P \perp\!\!\!\perp x \land \llbracket F \rrbracket \subseteq P \Rightarrow a \in P\big)$$

$\Rightarrow$: We assume $a \in \llbracket \exists x.\, F \rrbracket$ and take arbitrary but fixed $P \in Pred$ for which we assume

$$P \perp\!\!\!\perp x \tag{9}$$
$$\llbracket F \rrbracket \subseteq P \tag{10}$$

Our goal is to show $a \in P$. From $a \in \llbracket \exists x.\, F \rrbracket$, we know $a[x \mapsto v] \in \llbracket F \rrbracket$ for some $v \in Val$. From (10), we thus know $a[x \mapsto v] \in P$. From (9), we thus know $a[x \mapsto a(x)] \in P$. Since $a[x \mapsto a(x)] = a$, we thus know $a \in P$.

$\Leftarrow$: We assume

$$\forall P \in Pred.\, P \perp\!\!\!\perp x \wedge [\![ F ]\!] \subseteq P \Rightarrow a \in P \tag{11}$$

and prove $a \in [\![ \exists x.\, F ]\!]$. From (11) instantiated with $P := [\![ \exists x.\, F ]\!]$ and Propositions 3 and 2, it suffices to prove $[\![ F ]\!] \subseteq [\![ \exists x.\, F ]\!]$. Take arbitrary assignment $a_0 \in [\![ F ]\!]$. Since $a_0[x \mapsto a(x)] = a$, we thus have $a_0[x \mapsto v] \in [\![ F ]\!]$ for $v := a(x)$ and thus $a_0 \in [\![ \exists x.\, F ]\!]$. $\square$

## 3. Category Theory

In this section, we discuss those aspects of category theory that are relevant for the subsequent categorical formulation of our relational first-order logic.

### 3.1. Basic Notions

We begin with the basic notions of category theory.

**Definition 13** (Category). *A category $\mathcal{C}$ is a triple $\langle O, A, \circ \rangle$ of the following components:*

- *A class $O$ of elements called $\mathcal{C}$-*objects *or just* objects.
- *A class $A$ of elements called $\mathcal{C}$-*arrows *or just* arrows. *Each arrow has a* source *object and a* target *object from O; we write $f \colon a \to b$ to indicate that $f$ is an arrow with source $a$ and target $b$. We write $\mathcal{C}(a, b)$ to denote the class of all arrows of $A$ with source $a$ and target $b$ (called the* hom-class *of all arrows from $a$ to $b$). For every object $x$ in O, A contains an arrow $id_x \colon x \to x$ called the* identity *arrow for $x$.*
- *A composition—binary operation $\circ$ defined on arrows. For all arrows $f \colon a \to b$ and $g \colon b \to c$, we have $(g \circ f) \colon a \to c$. Furthermore, the composition satisfies the following axioms:*
  - *Associativity: $(h \circ g) \circ f = h \circ (g \circ f)$, for all arrows $f \colon a \to b$, $g \colon b \to c$, $h \colon c \to d$.*
  - *Identity: $id_b \circ f = f = f \circ id_a$, for all arrows $f \colon a \to b$.*

**Definition 14** (Isomorphism). *Let $\mathcal{C}$ be a category and $a, b$ be $\mathcal{C}$-objects $a, b$. Then, we have $a \simeq b$ (read: $a$ and $b$ are isomorphic) if there are $\mathcal{C}$-arrows $f \colon a \to b$ and $g \colon b \to a$, called isomorphisms, such that $g \circ f = id_a$ and $f \circ g = id_b$.*

**Definition 15** (Subcategory). *A category $\mathcal{C}$ is a subcategory of category $\mathcal{D}$ if every $\mathcal{C}$-object is also a $\mathcal{D}$-object, every $\mathcal{C}$-arrow is also a $\mathcal{D}$-arrow, every identity arrow in $\mathcal{C}$ is also an identity arrow in $\mathcal{D}$, and $g \circ_C f = g \circ_D f$ for all $\mathcal{C}$-arrows $f \colon a \to b$ and $g \colon b \to c$, where $\circ_C$ denotes the composition in $\mathcal{C}$ and $\circ_D$ denotes the composition in $\mathcal{D}$.*

### 3.2. Object Constructions

We are now introducing constructions of categorical objects that will subsequently play an important role in the categorical formulation of relational first-order logic.

**Definition 16** (Initial and Final Objects). *Let $\mathcal{C}$ be a category. A $\mathcal{C}$-object* $0$ is initial *if for every $\mathcal{C}$-object $a$ there exists exactly one arrow $0_a : 0 \to a$. A $\mathcal{C}$-object* $1$ is final *if for every $\mathcal{C}$-object $a$ there exists exactly one arrow $1_a : a \to 1$.*

The following diagram illustrates the arrows of an initial object 0 and a final object 1 with respect to an arbitrary object $a$:

$$\begin{tikzcd}
1 \\
a \arrow[u, dashed, "1_a"'] \\
0 \arrow[u, dashed, "0_a"']
\end{tikzcd}$$

This construction of initial/final objects is "universal" in the sense that it describes a class of entities (objects and accompanying arrows) that share a common property and picks from this class an entity whose characterizing property is the existence of exactly one arrow from/to every entity of this class. This defines the entity uniquely up to isomorphism. Further instances of such constructions will be given later.

**Definition 17** (Product and Coproduct). *Let $\mathcal{C}$ be a category. Then, the triple $\langle a \times b, \pi_1, \pi_2 \rangle$ is a product of $\mathcal{C}$-objects $a$ and $b$ if $a \times b$ is a $\mathcal{C}$-object, the product object, with arrows $\pi_1 \colon a \times b \to a$ and $\pi_2 \colon a \times b \to b$, the* projections, *such that for every triple $\langle c, f, g \rangle$ with $\mathcal{C}$-object $c$ and arrows $f \colon c \to a$ and $g \colon c \to b$ there exists exactly one arrow $\langle f, g \rangle \colon c \to a \times b$ such that the following diagram commutes:*

$$\begin{tikzcd}
a & a \times b \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & b \\
& c \arrow[ul, "f"] \arrow[u, dashed, "\langle f,g \rangle"'] \arrow[ur, "g"'] &
\end{tikzcd}$$

*Dually, the triple $(a + b, \iota_1, \iota_2)$ is a* coproduct *of $\mathcal{C}$-objects $a$ and $b$ if $a + b$ is a $\mathcal{C}$-object, the* coproduct object, *with arrows $\iota_1 \colon a \to a + b$ and $\iota_2 \colon b \to a + b$, the* injections, *such that, for every triple $\langle c, f, g \rangle$ with $\mathcal{C}$-object $c$ and arrows $f \colon a \to c$ and $g \colon b \to c$, there exists exactly one arrow $[f, g] \colon a + b \to c$ such that the following diagram commutes:*

$$\begin{tikzcd}
a \arrow[r, "\iota_1"] \arrow[dr, "f"'] & a + b \arrow[d, dashed, "{[f,g]}"] & b \arrow[l, "\iota_2"'] \arrow[dl, "g"] \\
& c &
\end{tikzcd}$$

The product and the coproduct are thus defined by universal constructions analogous to those of the final and the initial element, respectively; thus, products and coproducts are also uniquely defined up to isomorphism.

**Definition 18** (Product Arrow). *Let $\mathcal{C}$ be a category with products $\langle a_1 \times a_2, \pi_1, \pi_2 \rangle$ and $\langle b_1 \times b_2, \pi_1', \pi_2' \rangle$ and arrows $f \colon a_1 \to b_1$ and $g \colon a_2 \to b_2$, respectively. Then, the product arrow $f \times g \colon a_1 \times a_2 \to b_1 \times b_2$ is the arrow $\langle f \circ \pi_1, g \circ \pi_2 \rangle$.*

**Definition 19** (Exponential). *Let $\mathcal{C}$ be a category in which, for all $\mathcal{C}$-objects, there exists a product object. Then, the tuple $\langle b^a, eval_{a,b} \rangle$ is an* exponential *of $\mathcal{C}$-objects $a$ and $b$ if $b^a$ is a $\mathcal{C}$-object, the* exponential object, *with arrow $eval_{a,b}\colon b^a \times a \to b$, the* evaluation arrow, *such that for every $\mathcal{C}$-object $c$ with arrow $f\colon c \times a \to b$ there exists exactly one arrow $curry_f\colon c \to b^a$, the* currying arrow, *such that the following diagram commutes:*

$$\begin{tikzcd}
b^a \times a \arrow[r, "eval_{a,b}"] & b \\
c \times a \arrow[u, dashed, "curry_f \times id_a"] \arrow[ur, "f"'] &
\end{tikzcd}$$

Since the exponential is also defined by a universal construction, it is uniquely defined up to isomorphism.

## 3.3. Functors and Adjunction

Moving on from individual categories, we will now discuss some concepts that address relationships between categories.

**Definition 20** (Functor). *Let $\mathcal{C}$ and $\mathcal{D}$ be categories. A functor $F\colon \mathcal{C} \to \mathcal{D}$ is a map that takes every $\mathcal{C}$-object $a$ to a $\mathcal{D}$-object $F(a)$ and every $\mathcal{C}$-arrow $f\colon a \to b$ to a $\mathcal{D}$-arrow $F(f)\colon F(a) \to F(b)$ such that*

- *$F(id_a) = id_{F(a)}$ for every $\mathcal{C}$-object $a$, and*
- *$F(g \circ_C f) = F(g) \circ_D F(f)$ for all $\mathcal{C}$-arrows $f\colon a \to b$ and $g\colon b \to c$.*

**Definition 21** (Adjunction, Left, and Right Adjoint). *Let $\mathcal{C}$ and $\mathcal{D}$ be categories with functors $F\colon \mathcal{C} \to \mathcal{D}$ and $G\colon \mathcal{D} \to \mathcal{C}$. Then, we have $F \dashv G$ (read: $\langle F, G\rangle$ is an adjunction, $F$ is a left adjoint of $G$, $G$ is a right adjoint of $F$) if for every $\mathcal{C}$-object $a$ and $\mathcal{D}$-object $b$ the arrow classes $\mathcal{D}(F(a), b)$ and $\mathcal{C}(a, G(b))$ are isomorphic, i.e., there exists a bijection between them. This is equivalent to saying that, for every $\mathcal{C}$-object $a$ and $\mathcal{D}$-object $b$, there exist two surjective mappings $s_1\colon \mathcal{D}(F(a), b) \to \mathcal{C}(a, G(b))$ and $s_2\colon \mathcal{C}(a, G(b)) \to \mathcal{D}(F(a), b)$, i.e.,*

- *for every $\mathcal{D}$-arrow $g\colon F(a) \to b$ we have a $\mathcal{C}$-arrow $f\colon a \to G(b)$ with $s_2(f) = g$ and*
- *for every $\mathcal{C}$-arrow $f\colon a \to G(b)$, we have a $\mathcal{D}$-arrow $g\colon F(a) \to b$ with $s_1(g) = f$.*

**Note.** *This equivalence is a consequence of the Cantor–Schröder–Bernstein theorem which states that there exists a bijective function between sets $A$ and $B$ if there exist injective functions $f\colon A \to B$ and $g\colon B \to A$. This implies that such a bijective function also exists if there exist surjective functions $f'\colon A \to B$ and $g'\colon B \to A$ because, from these, we can define the injective functions $f(a) :=$ such $b$. $g'(b) = a$ and $g(b) :=$ such $a$. $f'(a) = b$. While the theorem has been formulated for sets, it can also be generalized to classes.*

*The above formulation will become handy in proving that two functors represent an adjunction.*

**Proposition 7** (Equivalence of Adjunctions and Universals). *Let $\mathcal{C}$ and $\mathcal{D}$ be categories with functors $F\colon \mathcal{C} \to \mathcal{D}$ and $G\colon \mathcal{D} \to \mathcal{C}$. Then, the condition $F \dashv G$ is equivalent to each of the following two conditions:*

1. *For every $\mathcal{C}$-object $a$, there is a $\mathcal{C}$-arrow $u\colon a \to G(F(a))$, the "universal arrow", such that, for every $\mathcal{D}$-object $b$ and $\mathcal{C}$-arrow $f\colon a \to G(b)$, there exists a $\mathcal{D}$-arrow $g_{b,f}\colon F(a) \to b$:*

$$\begin{tikzcd}
a \arrow[r, "u"] \arrow[dr, "f"'] & G(F(a)) \arrow[d, "G(g_{b,f})"] & F(a) \arrow[d, dashed, "g_{b,f}"] \\
& G(b) & b
\end{tikzcd}$$

2. *For every $\mathcal{D}$-object $b$, there is a $\mathcal{C}$-arrow $v\colon F(G(b)) \to b$, the "couniversal arrow", such that, for every $\mathcal{C}$-object $a$ and $\mathcal{D}$-arrow $g\colon F(a) \to b$, there is a $\mathcal{C}$-arrow $f_{a,g}\colon a \to G(b)$:*

$$\begin{tikzcd}
G(b) & b & F(G(b)) \arrow[l, "v"'] \\
a \arrow[u, dashed, "f_{a,g}"'] & & F(a) \arrow[ul, "g"] \arrow[u, "F(f_{a,g})"']
\end{tikzcd}$$

**Proof.** See the proof of Propositions 6 and 7 in [12]. □

### 3.4. Object Constructions by Adjunction

We conclude this section by demonstrating that the previously described object conjunctions can be also considered as applications of functors that are determined as left respectively right adjoints to certain basic functors.

**Proposition 8** (Initial and Final Object by Adjunction). *Let $\mathbf{1}$ be the "singleton" category with a single object $*$ (and consequently a single arrow $id_*: * \to *$); this category is uniquely defined up to isomorphism. Let $\mathcal{C}$ be a category with the constant functor $C: \mathcal{C} \to \mathbf{1}$; in addition, this functor is uniquely defined up to isomorphism. Then, the following holds:*

- *Let $\mathcal{C}$-object 0 be initial and the "initial object functor" $I_0: \mathbf{1} \to \mathcal{C}$ be defined by $I_0(*) := 0$ and $I_0(id_*) := id_0$. Then, we have $I_0 \dashv C$, i.e., the initial object functor is a left adjoint of the constant functor.*
- *Let $\mathcal{C}$-object 1 be final and the "final object functor" $F_1: \mathbf{1} \to \mathcal{C}$ be defined by $F_1(*) := 1$ and $F_1(id_*) := id_1$. Then, we have $C \dashv F_1$, i.e., the final object functor is a right adjoint of the constant functor.*

**Proof.** For showing the first statement, we take the initial object 0 with initial object functor $I_0$. We show $I_0 \dashv C$, i.e., that $\mathcal{C}(I_0(*), a)$ and $\mathbf{1}(*, C(a))$ are isomorphic, for arbitrary $\mathcal{C}$-object $a$. This follows from $\mathbf{1}(*, C(a)) = \mathbf{1}(*, *)$, $\mathcal{C}(I_0(*), a) = \mathcal{C}(0, a)$, and the fact that there exists exactly one $\mathbf{1}$-arrow $id_*: * \to *$ and, since 0 is initial, exactly one $\mathcal{C}$-arrow $f: 0 \to a$.

For showing the second statement, we take the final object 1 with final functor $F_1$. We prove $C \dashv F_1$, i.e., that $\mathbf{1}(C(a), *)$ and $\mathcal{C}(a, F_1(*))$ are isomorphic, for arbitrary $\mathcal{C}$-object $a$. This follows from $\mathbf{1}(C(a), *) = \mathbf{1}(*, *)$, $\mathcal{C}(a, F_1(*)) = \mathcal{C}(a, 1)$, and the fact that there exists exactly one $\mathbf{1}$-arrow $id_*: * \to *$ and, since 1 is final, exactly one $\mathcal{C}$-arrow $f: a \to 1$. □

**Proposition 9** (Product and Coproduct by Adjunction). *Let $\mathcal{C}$ be a category. Let the "product category" $\mathcal{C} \times \mathcal{C}$ be the category whose objects $(a, b)$ are pairs of $\mathcal{C}$-objects $a$ and $b$, whose arrows $(f, g) : (a, c) \to (b, d)$ are pairs of $\mathcal{C}$-arrows $f: a \to b$ and $g: c \to d$, where the identity arrows are pairs of identity arrows, and where composition is component-wise composition. Let the "diagonal functor" $\Delta: \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ be defined by $\Delta(a) = (a, a)$ for every $\mathcal{C}$-object $a$ and $\Delta(f) = (f, f)$ for every $\mathcal{C}$-arrow $f: a \to b$. Then, the following holds:*

- *Assume that every pair of $\mathcal{C}$-objects $a$ and $b$ has a product $a \times b$ and let the "product functor" $P: \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ be defined by $P(a, b) := a \times b$ . Then, we have $\Delta \dashv P$, i.e., the product functor is a right adjoint of the diagonal functor.*
- *Assume that every pair of $\mathcal{C}$-objects $a$ and $b$ has a coproduct $a + b$ and let the "coproduct functor" $C: \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ be defined by $C(a, b) := a + b$. Then, we have $C \dashv \Delta$, i.e., the coproduct functor is a left adjoint of the diagonal functor.*

**Proof.** For showing the first statement, we take arbitrary category $\mathcal{C}$ and functor $P$ satisfying the stated assumption. We show $\Delta \dashv P$, i.e., that, for arbitrary $\mathcal{C}$-objects $p, a, b$, the arrow classes $(\mathcal{C} \times \mathcal{C})(\Delta(p), (a, b))$ and $\mathcal{C}(p, P(a, b))$ are isomorphic. Since $\Delta(p) = (p, p)$ and $P(a, b) = a \times b$, it suffices to find surjections $s_1: (\mathcal{C} \times \mathcal{C})((p, p), (a, b)) \to \mathcal{C}(p, a \times b)$ and $s_2: \mathcal{C}(p, a \times b) \to (\mathcal{C} \times \mathcal{C})((p, p), (a, b))$. First, we define $s_1(f, g) := \langle f, g \rangle$ where $\langle f, g \rangle: p \to a \times b$ is the unique $\mathcal{C}$-arrow given to us by Definition 17 with property $f = \pi_1 \circ \langle f, g \rangle$ and $g = \pi_2 \circ \langle f, g \rangle$. Now, we show that, for every $\mathcal{C}$-arrow $h: p \to a \times b$, there exist some $\mathcal{C}$-arrows $f: p \to a$ and $g: p \to b$ with $s_1(f, g) = h$. We take $f := \pi_1 \circ h$ and $g := \pi_2 \circ h$. Due to the uniqueness of $\langle f, g \rangle$, the equalities $f = \pi_1 \circ h$ and $g = \pi_2 \circ h$ imply

$h = \langle f, g\rangle$ and thus $s_1(f, g) = h$. Second, we define $s_2(h) := (\pi_1 \circ h, \pi_2 \circ h)$. Now, we show that, for every $(\mathcal{C} \times \mathcal{C})$-arrow $(f, g)\colon (p, p) \to (a, b)$, i.e., for all $\mathcal{C}$-arrows $f\colon p \to a$ and $g\colon p \to b$, there exists some $\mathcal{C}$-arrow $h\colon p \to a \times b$ with $s_2(h) = (f, g)$, i.e., $\pi_1 \circ h = f$ and $\pi_2 \circ h = g$. Definition 17 can be used to define $h$.

For showing the second statement, we take arbitrary category $\mathcal{C}$ and functor $C$ satisfying the stated assumption. We prove $C \dashv \Delta$, i.e., that, for arbitrary $\mathcal{C}$-objects $a, b, c$, the arrow classes $\mathcal{C}(C(a, b), c)$ and $(\mathcal{C} \times \mathcal{C})((a, b), \Delta(c))$ are isomorphic. Since $C(a, b) = a + b$ and $\Delta(c) = (c, c)$, it suffices to find surjections $s_1\colon \mathcal{C}(a + b, c) \to (\mathcal{C} \times \mathcal{C})((a, b),(c, c))$ and $s_2\colon (\mathcal{C} \times \mathcal{C})((a, b),(c, c)) \to \mathcal{C}(a + b, c)$. First, we define $s_1(h) := (h \circ \iota_1, h \circ \iota_2)$. Now, we prove that for every $(\mathcal{C} \times \mathcal{C})$-arrow $(f, g)\colon (a, b) \to (c, c)$, i.e., for all $\mathcal{C}$-arrows $f\colon a \to c$ and $g\colon b \to c$, there exists some $\mathcal{C}$-arrow $h\colon a + b \to c$ with $s_1(h) = (f, g)$, i.e., $h \circ \iota_1 = f$ and $h \circ \iota_2 = g$. Definition 17 can be used to define $h$. Second, we define $s_2(f, g) := [f, g]$ where $[f, g]\colon a + b \to c$ is the unique $\mathcal{C}$-arrow given to us by Definition 17 with property $f = [f, g] \circ \iota_1$ and $g = [f, g] \circ \iota_2$. Now, we show that, for every $\mathcal{C}$-arrow $h\colon a + b \to c$, there exist some $\mathcal{C}$-arrows $f\colon a \to c$ and $g\colon b \to c$ with $s_2(f, g) = h$. We take $f := h \circ \iota_1$ and $g := h \circ \iota_2$. Due to the uniqueness of $[f, g]$, the equalities $f = h \circ \iota_1$ and $g = h \circ \iota_2$ imply $h = [f, g]$ and thus $s_2(f, g) = h$. □

**Proposition 10** (Exponential by Adjunction)**.** *Let $\mathcal{C}$ be a category in which, for every pair of $\mathcal{C}$-objects $a$ and $b$, there exists a product object $b \times a$ and an exponential object $b^a$. For every $\mathcal{C}$-object $a$, let the "(unary) product functor" $P_a\colon \mathcal{C} \to \mathcal{C}$ be defined by $P_a(b) := b \times a$ and the "(unary) exponential functor" $E_a\colon \mathcal{C} \to \mathcal{C}$ be defined by $E_a(b) := b^a$. Then, we have $P_a \dashv E_a$, i.e., the exponential functor is a right adjoint of the product functor.*

**Proof.** We take arbitrary category $\mathcal{C}$, $\mathcal{C}$-object $a$, and functors $P_a$ and $E_a$ satisfying the assumption. We show $P_a \dashv E_a$, i.e., that, for arbitrary $\mathcal{C}$-objects $b, c$, the arrow classes $\mathcal{C}(P_a(c), b)$ and $\mathcal{C}(c, E_a(b))$ are isomorphic. Since $P_a(b) = b \times a$ and $E_a(b) = b^a$, it suffices to find surjections $s_1\colon \mathcal{C}(c \times a, b) \to \mathcal{C}(c, b^a)$ and $s_2\colon \mathcal{C}(c, b^a) \to \mathcal{C}(c \times a, b)$. First, we define $s_1(f) := curry_f$. Now, we show that, for every $\mathcal{C}$-arrow $g\colon c \to b^a$, there exists some $\mathcal{C}$-arrow $f\colon c \times a \to b$ with $s_1(f) = g$, i.e., $curry_f = g$. We define $f := eval_{a,b} \circ (g \times id_a)$ and show $curry_f = g$. From the definition of $f$, we know that the $\mathcal{C}$-arrow $g\colon c \to b^a$ satisfies the equality $f = eval_{a,b} \circ (g \times id_a)$. However, Definition 19 implies that the only such $\mathcal{C}$-arrow is $curry_f$; thus, $curry_f = g$. Second, we define $s_2(g) := eval_{a,b} \circ (g \times id_a)$. Now, we show that, for every $\mathcal{C}$-arrow $f\colon c \times a \to b$, there exists some $\mathcal{C}$-arrow $g\colon c \to b^a$ with $s_2(g) = f$, i.e., $eval_{a,b} \circ (g \times id_a) = f$. We define $g := curry_f$ from which Definition 19 proves the goal. □

We are now ready to discuss the central aspects of categorical logic.

## 4. A Categorical Semantics

Based on the concepts introduced in the previous sections, this section elaborates a categorical semantics of our relational version of first-order logic. We advise the reader to consult Figure 1 to grasp the overall framework and the relationship between its various categories and functors.

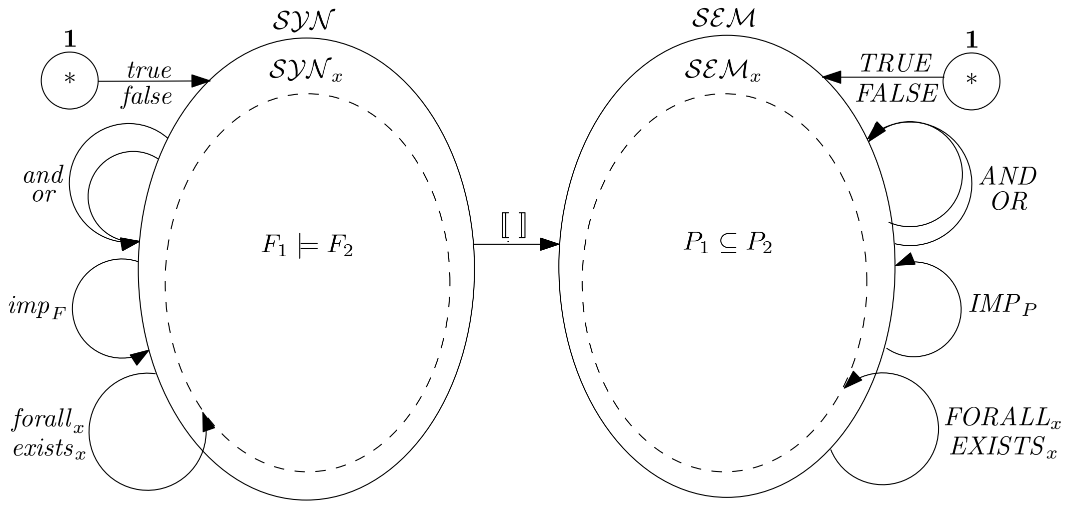


**Figure 1.** A Categorical Semantics of Relational First-Order Logic

## *4.1. Syntactic Category and Formula Functors*

We start by introducing the "syntactic category" $\mathcal{SYN} = \langle For, A, \circ \rangle$ as follows:

- The objects of this category are the formulas in the set *For* which was introduced in Definition 6.
- The arrow class $A$ consists of all pairs $\langle F_1, F_2 \rangle$ of formulas $F_1, F_2$ for which $F_1 \models F_2$ holds, i.e., for which $F_2$ is a logical consequence of $F_1$, as described in Definition 8. The source object of such an arrow is $F_1$, and its target object is $F_2$. The existence of an arrow $f\colon F_1 \to F_2$ thus indicates $F_1 \models F_2$. The identity $id_F\colon F \to F$ indicates the fact $F \models F$.
- The composition $\circ$ denotes relational composition: for all arrows $f\colon F_1 \to F_2$ and $g\colon F_2 \to F_3$, and the existence of the arrow $(g \circ f)\colon F_1 \to F_3$ indicates the transitivity of the relation $\models$.

For every variable $x$, $\mathcal{SYN}_x$ is that subcategory of $\mathcal{SYN}$ whose objects are formulas whose semantics are independent of $x$ (see Definition 4).

For reasons explained below, we will exclude from the syntactic category negations and equivalences, i.e., formulas of form $(\neg F)$ and $(F_1 \leftrightarrow F_2)$. We may do so by considering them as the following syntactic shortcuts:

$$(\neg F) \equiv (F \to \bot),$$
$$(F_1 \leftrightarrow F_2) \equiv (F_1 \to F_2) \land (F_2 \to F_1).$$

The validity of these shortcuts can be easily shown by proving the corresponding logical equivalences. Consequently, negations and equivalences need subsequently not be considered any more and their semantics need not be explicitly defined.

For the other kinds of formulas, we introduce the following (families of) "formula functors" where **1** is the "singleton" category with a single object $*$ (see Proposition 8):

$$\begin{aligned}
true\colon & \mathbf{1} \to \mathcal{SYN} \\
false\colon & \mathbf{1} \to \mathcal{SYN} \\
and\colon & \mathcal{SYN} \times \mathcal{SYN} \to \mathcal{SYN} \\
or\colon & \mathcal{SYN} \times \mathcal{SYN} \to \mathcal{SYN} \\
imp_{F \in For}\colon & \mathcal{SYN} \to \mathcal{SYN} \\
forall_{x \in Var}\colon & \mathcal{SYN} \to \mathcal{SYN}_x \\
exists_{x \in Var}\colon & \mathcal{SYN} \to \mathcal{SYN}_x
\end{aligned}$$

These functors map formulas to formulas, and logical consequences to logical consequences. The formula mappings are naturally defined as follows:

$$\begin{aligned}
\mathit{true}(*) &:= \top \\
\mathit{false}(*) &:= \bot \\
\mathit{and}(F_1, F_2) &:= F_1 \wedge F_2 \\
\mathit{or}(F_1, F_2) &:= F_1 \vee F_2 \\
\mathit{imp}_{F_1}(F_2) &:= F_1 \rightarrow F_2 \\
\mathit{forall}_x(F) &:= \forall x.\ F \\
\mathit{exists}_x(F) &:= \exists x.\ F
\end{aligned}$$

As for the mapping of consequences, we notice that all functors are *covariant* in their $\mathcal{SYN}$ arguments. It is exactly for this reason that negation and equivalence (which do not allow covariance in their arguments) are not modeled as formula functors and that implication (which is only covariant in its second argument) is not modeled by a binary functor but by a family of unary functors.

Thus, we have for all formulas $F, F_1, F_2, G, G_1, G_2$ and every variable $x$ the following (easy to prove) properties:

$$\begin{aligned}
&(F_1 \models G_1) \wedge (F_2 \models G_2) \Rightarrow \mathit{and}(F_1, F_2) \models \mathit{and}(G_1, G_2) \\
&(F_1 \models G_1) \wedge (F_2 \models G_2) \Rightarrow \mathit{or}(F_1, F_2) \models \mathit{or}(G_1, G_2) \\
&(F_2 \models G_2) \Rightarrow \mathit{imp}_{F_1}(F_2) \models \mathit{imp}_{F_1}(G_2) \\
&(F \models G) \Rightarrow \mathit{forall}_x(F) \models \mathit{forall}_x(G) \\
&(F \models G) \Rightarrow \mathit{exists}_x(F) \models \mathit{exists}_x(G)
\end{aligned}$$

Therefore, the object maps of these functors naturally induce the necessary logical consequences.

## 4.2. Semantic Category and Predicate Functors

Next, we introduce the "semantic category" $\mathcal{SEM} = \langle \mathit{Pred}, B, \circ \rangle$ as follows:

- The objects of this category are the predicates in the set *Pred* which was introduced in Definition 5 (thus $\mathcal{SEM}$-objects are relations, i.e., sets).
- The arrow class $B$ consists of all pairs $\langle P_1, P_2 \rangle$ of predicates $P_1, P_2$ for which $P_1 \subseteq P_2$ holds, i.e., for which $P_1$ is a subset of $P_2$. The source object of such an arrow is $P_1$, its target object is $P_2$. The existence of an arrow $f\colon P_1 \rightarrow P_2$ thus indicates $P_1 \subseteq P_2$. The identity $id_P\colon P \rightarrow P$ indicates the fact $P \subseteq P$.
- The composition $\circ$ denotes relational composition: for all arrows $f\colon P_1 \rightarrow P_2$ and $g\colon P_2 \rightarrow P_3$, the existence of the arrow $(g \circ f)\colon P_1 \rightarrow P_3$ indicates the transitivity of the relation $\subseteq$.

For every variable $x$, $\mathcal{SEM}_x$ is the subcategory of $\mathcal{SEM}$ whose objects are predicates that are independent of $x$ (see Definition 4).

Corresponding to the various kinds of formula constructions, we will have the following "predicate functors" (respectively families of functors):

$$\begin{aligned} TRUE &\colon \mathbf{1} \to \mathcal{SEM} \\ FALSE &\colon \mathbf{1} \to \mathcal{SEM} \\ AND &\colon \mathcal{SEM} \times \mathcal{SEM} \to \mathcal{SEM} \\ OR &\colon \mathcal{SEM} \times \mathcal{SEM} \to \mathcal{SEM} \\ IMP_{P \in Pred} &\colon \mathcal{SEM} \to \mathcal{SEM} \\ FORALL_{x \in Var} &\colon \mathcal{SEM} \to \mathcal{SEM}_x \\ EXISTS_{x \in Var} &\colon \mathcal{SEM} \to \mathcal{SEM}_x \end{aligned}$$

These functors map predicates to predicates and subset relations to subset relations (their detailed definitions will be given later). As we will see, these functors are covariant in their $\mathcal{SEM}$-arguments, i.e., we have for all predicates $P, P_1, P_2, Q, Q_1, Q_2$ and every variable $x$ the following properties:

$$\begin{aligned} &(P_1 \subseteq Q_1) \wedge (P_2 \subseteq Q_2) \Rightarrow AND(P_1, P_2) \subseteq AND(Q_1, Q_2) \\ &(P_1 \subseteq Q_1) \wedge (P_2 \subseteq Q_2) \Rightarrow OR(P_1, P_2) \subseteq OR(Q_1, Q_2) \\ &(P_2 \subseteq Q_2) \Rightarrow IMP_{P_1}(P_2) \subseteq IMP_{P_1}(Q_2) \\ &(P \subseteq Q) \Rightarrow FORALL_x(P) \subseteq FORALL_x(Q) \\ &(P \subseteq Q) \Rightarrow EXISTS_x(P) \subseteq EXISTS_x(Q) \end{aligned}$$

Therefore, the object maps of these functors (defined by the respective predicate operations) naturally induce appropriate arrow maps (the corresponding subset relations).

### *4.3. The Semantic Functor*

Now, we introduce the "semantic functor"

$$[\![\ ]\!] \colon \mathcal{SYN} \to \mathcal{SEM}$$

defined as follows:

- For every $\mathcal{SYN}$-object $F$, i.e., formula $F$, $[\![ F ]\!]$ denotes the semantics of $F$ as defined in Definition 7, which according to Proposition 3 is a predicate, i.e., indeed a $\mathcal{SEM}$-object.
- For every $\mathcal{SYN}$-arrow $f \colon F_1 \to F_2$, i.e., every pair of formulas $F_1$ and $F_2$ with $F_1 \models F_2$, we have the $\mathcal{SEM}$-arrow $[\![ f ]\!] \colon [\![ F_1 ]\!] \to [\![ F_2 ]\!]$, i.e., the fact $[\![ F_1 ]\!] \subseteq [\![ F_2 ]\!]$, which is a direct consequence of Definition 10 which introduces the $\models$ relation.

This semantic functor establishes the relationship between the previously introduced formula functors and predicate functors by the following identities on $\mathcal{SEM}$-objects, i.e., predicate identities that will hold for all formulas $F, F_1, F_2$ and every variable $x$:

$$\begin{aligned} &[\![ true(*) ]\!] = TRUE(*) \\ &[\![ false(*) ]\!] = FALSE(*) \\ &[\![ and(F_1, F_2) ]\!] = AND([\![ F_1 ]\!], [\![ F_2 ]\!]) \\ &[\![ or(F_1, F_2) ]\!] = OR([\![ F_1 ]\!], [\![ F_2 ]\!]) \\ &[\![ imp_{F_1}(F_2) ]\!] = IMP_{[\![ F_1 ]\!]}([\![ F_2 ]\!]) \\ &[\![ forall_x(F) ]\!] = FORALL_x([\![ F ]\!]) \\ &[\![ exists_x(F) ]\!] = EXISTS_x([\![ F ]\!]) \end{aligned}$$

### *4.4. Categorical Semantics of First-Order Relational Logic*

We are now going to elaborate in detail the semantic functors from which all of the above can be shown; this elaboration is inspired from and indeed directly derived from the well-known logical inference rules of first-order logic. The resulting definitions are based on the categorical notions introduced in Section 3, i.e., final and initial objects, products and coproducts, exponentials, and left and right adjoints, respectively. This gives us for every logical operation a "universal" definition of its semantics. Nevertheless, this semantics is also "constructive" in the sense that it is explicitly defined from well-known set-theoretic operations.

#### 4.4.1. Logical Constants

The role of the logical constants in reasoning is exhibited by the following two "rules" which follow directly from Definition 8 (these rules are propositions that are valid for every formula $F$; they mimic the corresponding inference rules of first-order logic):

$$F \models \top \qquad \bot \models F$$

In other words, $\top$ is a logical consequence of every formula $F$, i.e., $\top$ is the "weakest" formula. Dually, every formula $F$ is a logical consequence of $\bot$, i.e., $\bot$ is the "strongest" formula. This implies that $true(*) = \top$ is the final object of category $\mathcal{SYN}$ and $false(*) = \bot$ is its initial one (see Definition 16). Then, Proposition 8 implies $C_{\mathcal{SYN}} \dashv true$ and $false \dashv C_{\mathcal{SYN}}$, i.e., functor *true* is the right adjoint of the constant functor $C_{\mathcal{SYN}} \colon \mathcal{SYN} \to \mathbf{1}$ while functor *false* is its left one.

Correspondingly, $TRUE(*)$ is the final object of category $\mathcal{SEM}$ (the "weakest" predicate, i.e., the predicate which is a superset of every predicate) and $FALSE(*)$ is its initial object (the "strongest" predicate, i.e., the predicate which is a subset of every predicate). By Proposition 8, we then have $C_{\mathcal{SEM}} \dashv TRUE$ and $FALSE \dashv C_{\mathcal{SEM}}$, i.e., functor *TRUE* is the right adjoint of the constant functor $C_{\mathcal{SEM}} \colon \mathcal{SEM} \to \mathbf{1}$ while functor *FALSE* is its left one.

Therefore, corresponding to the above rules for formulas, we have the following rules for every predicate $P$:

$$P \subseteq TRUE(*) \qquad FALSE(*) \subseteq P$$

Since final and initial objects are unique, these rules actually represent implicit but unique definitions of $TRUE(*)$ and $FALSE(*)$ which can be explicitly written as

$$TRUE(*) := \bigcup\{P \mid P \in Pred\} \qquad FALSE(*) := \bigcap\{P \mid P \in Pred\}$$

i.e., $TRUE(*)$ is the union of all predicates and $FALSE(*)$ is their intersection. Thus, we have derived alternative characterizations $[\![\top]\!] = TRUE(*)$ and $[\![\bot]\!] = FALSE(*)$ that are both constructive and universal (Proposition 5 gives us $[\![\top]\!] = Ass$ and $[\![\bot]\!] = \emptyset$ from which it is easy to verify these equalities).

#### 4.4.2. Conjunction and Disjunction

The role of conjunction in reasoning is exhibited by the following rules for arbitrary formulas $F_1, F_2, F$ (the first two ones mimic the logical inference rules of "elimination", and the last one mimics the inference rule of "introduction"):

$$\begin{gathered} F_1 \wedge F_2 \models F_1 \\ F_1 \wedge F_2 \models F_2 \\ (F \models F_1) \wedge (F \models F_2) \Rightarrow (F \models F_1 \wedge F_2) \end{gathered}$$

Dually, we have the following rules for disjunction:

$$
\begin{gathered}
F_1 \models F_1 \vee F_2 \\
F_2 \models F_1 \vee F_2 \\
(F_1 \models F) \wedge (F_2 \models F) \Rightarrow (F_1 \vee F_2 \models F)
\end{gathered}
$$

These rules (whose soundness can be established with the help of Definition 8) state that $(F_1 \wedge F_2)$ is the "weakest" formula $F$ for which both $(F \models F_1)$ and $(F \models F_2)$ hold and that $(F_1 \vee F_2)$ is the "strongest" formula $F$ for which both $(F_1 \models F)$ and $(F_2 \models F)$ hold. Thus, $\mathit{and}(F_1, F_2) = (F_1 \wedge F_2)$ is the *product* of the $\mathcal{SYN}$-objects $F_1$ and $F_2$ and $\mathit{or}(F_1, F_2) = (F_1 \vee F_2)$ is their *coproduct* (see Definition 17). Furthermore, by Proposition 9, we have $\Delta_{\mathcal{SYN}} \dashv \mathit{and}$ and $\mathit{or} \dashv \Delta_{\mathcal{SYN}}$ i.e., functor *and* is the right adjoint of the diagonal functor $\Delta_{\mathcal{SYN}} \colon \mathcal{SYN} \to \mathcal{SYN} \times \mathcal{SYN}$ while functor *or* is its left one.

Correspondingly $\mathit{AND}(P_1, P_2)$ is the product of the $\mathcal{SEM}$-objects $P_1$ and $P_2$ (the "weakest" predicate $P$ for which both $(P \subseteq P_1)$ and $(P \subseteq P_2)$ hold) and $\mathit{OR}(P_1, P_2)$ is their coproduct (the "strongest" predicate $P$ for which $(P_1 \subseteq P)$ and $(P_2 \subseteq P)$ hold). By Proposition 9, we then have $\Delta_{\mathcal{SEM}} \dashv \mathit{AND}$ and $\mathit{OR} \dashv \Delta_{\mathcal{SEM}}$ i.e., functor *AND* is the right adjoint of the diagonal functor $\Delta_{\mathcal{SEM}} \colon \mathcal{SEM} \to \mathcal{SEM} \times \mathcal{SEM}$, while functor *OR* is its left one.

Thus, we have, corresponding to the rules for formulas, the following rules for all predicates $P_1, P_2, P$:

$$
\begin{gathered}
\mathit{AND}(P_1, P_2) \subseteq P_1 \\
\mathit{AND}(P_1, P_2) \subseteq P_2 \\
(P \subseteq P_1) \wedge (P \subseteq P_2) \Rightarrow (P \subseteq \mathit{AND}(P_1, P_2))
\end{gathered}
$$

Dually, we have

$$
\begin{gathered}
P_1 \subseteq \mathit{OR}(P_1, P_2) \\
P_2 \subseteq \mathit{OR}(P_1, P_2) \\
(P_1 \subseteq P) \wedge (P_2 \subseteq P) \Rightarrow (\mathit{OR}(P_1, P_2) \subseteq P)
\end{gathered}
$$

Since products and coproducts are uniquely defined, these rules actually represent implicit but unique definitions of $\mathit{AND}(P_1, P_2)$ and $\mathit{OR}(P_1, P_2)$ which can be explicitly written as follows:

$$
\begin{aligned}
\mathit{AND}(P_1, P_2) &:= \bigcup \{P \in \mathit{Pred} \mid P \subseteq P_1 \wedge P \subseteq P_2\} \\
\mathit{OR}(P_1, P_2) &:= \bigcap \{P \in \mathit{Pred} \mid P_1 \subseteq P \wedge P_2 \subseteq P\}
\end{aligned}
$$

This gives us alternative characterizations $[\![ F_1 \wedge F_2 ]\!] = [\![ F_1 ]\!] \cup [\![ F_2 ]\!] = \mathit{AND}([\![ F_1 ]\!], [\![ F_2 ]\!])$ and $[\![ F_1 \vee F_2 ]\!] = [\![ F_1 ]\!] \cup [\![ F_2 ]\!] = \mathit{OR}([\![ F_1 ]\!], [\![ F_2 ]\!])$ that are both constructive and universal (Proposition 5 implies $[\![ F_1 \wedge F_2 ]\!] = [\![ F_1 ]\!] \cap [\![ F_2 ]\!]$ and $[\![ F_1 \vee F_2 ]\!] = [\![ F_1 ]\!] \cup [\![ F_2 ]\!]$ from which it is not difficult to verify these equalities).

### 4.4.3. Implication

The role of implication in reasoning is exhibited by the following rules for arbitrary formulas $F_1, F_2, F$ (the first rule mimics the logical inference rules of "implication elimination" or "modus ponens", the last one mimics the inference rule of "implication introduction"):

$$
\begin{gathered}
(F_1 \to F_2) \wedge F_1 \models F_2 \\
(F \wedge F_1 \models F_2) \Rightarrow (F \models F_1 \to F_2)
\end{gathered}
$$

These rules (whose soundness can be established with the help of Definition 8) state that $(F_1 \to F_2)$ is the "weakest" formula $F$ for which $(F \wedge F_1 \models F_2)$ holds. Thus, $\mathit{imp}_{F_1}(F_2) = (F_1 \to F_2)$ is the *exponential* of the $\mathcal{SYN}$-objects $F_1$ and $F_2$ (see Definition 19). Proposition 10 then gives us $\mathit{and}_{F_1} \dashv \mathit{imp}_{F_1}$, i.e., functor $\mathit{imp}_{F_1}$ is the right adjoint of the unary conjunction functor $\mathit{and}_{F_1} \colon \mathcal{SYN} \to \mathcal{SYN} \times \mathcal{SYN}$ with object map $\mathit{and}_{F_1}(F_2) := \mathit{and}(F_1, F_2) = F_1 \wedge F_2$.

Correspondingly, $IMP_{P_1}(P_2)$ is the product of the $\mathcal{SEM}$-objects $P_1$ and $P_2$ (the "weakest" predicate $P$ for which $(P \cap P_1 \subseteq P_2)$ holds; Proposition 10 then gives us $AND_{P_1} \dashv IMP_{P_1}$, i.e., functor $IMP_{P_1}$ is the right adjoint of the unary functor $AND_{P_1} \colon \mathcal{SEM} \to \mathcal{SEM} \times \mathcal{SEM}$ with object map $AND_{P_1}(P_2) := AND(P_1, P_2) = P_1 \cup P_2$.

Thus, corresponding to above rules for formulas, we have the following rules for all predicates $P_1, P_2, P$:

$$IMP_{P_1}(P_2) \cap P_1 \subseteq P_2$$
$$(P \cap P_1 \subseteq P_2) \Rightarrow (P \subseteq IMP_{P_1}(P_2))$$

Since exponentials are uniquely defined, these rules represent an implicit but unique definition of $IMP_{P_1}(P_2)$ which can be explicitly written as follows:

$$IMP_{P_1}(P_2) := \bigcup \{P \in Pred \mid P \cap P_1 \subseteq P_2\}$$

This gives us an alternative characterization $[\![ F_1 \to F_2 ]\!] = IMP_{[\![ F_1 ]\!]}([\![ F_2 ]\!])$ that is both constructive and universal (Proposition 5 implies $[\![ F_1 \wedge F_2 ]\!] = \overline{[\![ F_1 ]\!]} \cup [\![ F_2 ]\!]$ from which it is possible to verify this equality).

### 4.4.4. Universal and Existential Quantification

The role of universal quantification in reasoning is exhibited by the following rules for arbitrary formulas $F, G$ provided that the semantics $[\![ G ]\!]$ of $G$ *do not depend* on $x$ (see Definition 4):

$$\forall x.\, F \models F$$
$$(G \models F) \Rightarrow (G \models \forall x.\, F)$$

The first rule mimics the logical inference rule of "universal elimination", the second one mimics the inference rule of "universal introduction" (except that our version of first-order logic does not involve terms and variables and thus copes without variable substitutions). This pair of rules in a nutshell yields that $(\forall x.\, F)$ is the "weakest" formula $G$ from which $F$ is a logical consequence and whose semantics *do not depend* on $x$. Dually, we have for existential quantification the following pair of rules:

$$F \models \exists x.\, F$$
$$(F \models G) \Rightarrow (\exists x.\, F \models G)$$

These rules state that $(\exists x.\, F)$ is the "strongest" formula $G$ that is a logical consequence of $F$ and whose semantics $[\![ G ]\!]$ *does not depend on* $x$.

We are now going to derive appropriate categorical characterizations of the corresponding functors $forall_{x \in Var} \colon \mathcal{SYN} \to \mathcal{SYN}_x$ and $exists_{x \in Var} \colon \mathcal{SYN} \to \mathcal{SYN}_x$ from the category $\mathcal{SYN}$ of all formulas to the subcategory $\mathcal{SYN}_x$ of all those formulas whose semantics do not depend on $x$. For this, we may notice that, from above rules, the relations $(G \models F)$ and $(F \models G)$ involve two kinds of relations, a more general relation $F$ that may depend on $x$ and a more special relation $G$ that is independent of $x$. In order to bring all relations to the "same level", we introduce a syntactic "injection" functor $I_x \colon \mathcal{SYN}_x \to \mathcal{SYN}$ whose maps are just identities, i.e., $I_x(G) = G$ and $I_x(f \colon F \to G) = f \colon F \to G$. This allows us to express above rules as

$$I_x(forall_x(F)) \models F$$
$$(I_x(G) \models F) \Rightarrow (G \models forall_x(F))$$

and dually

$$F \models I_x(exists_x(F))$$
$$(F \models I_x(G)) \Rightarrow (exists_x(F) \models G)$$

Now, the first set of rules matches the assumptions of the second part of Proposition 7 for $F := I_x$ and $G := \mathit{forall}_x$ (considering that the satisfaction relation $\models$ denotes the existence of an arrow in categories $\mathcal{SYN}$, respectively $\mathcal{SYN}_x$); thus, we have $I_x \dashv \mathit{forall}_x$. Likewise, the second set of rules matches the assumptions of the first part of that proposition for $F := \mathit{exists}_x$ and $G := I_x$; thus, we have $\mathit{exists}_x \dashv I_x$. Summarizing, the universal functor $\mathit{forall}_x$ is the right adjoint of the injection functor $I_x$ while the existential functor $\mathit{exists}_x$ is its left adjoint.

These considerations can be easily transferred to categorical characterizations of the corresponding functors $\mathit{FORALL}_{x \in \mathit{Var}} \colon \mathcal{SEM} \to \mathcal{SEM}_x$ and $\mathit{EXISTS}_{x \in \mathit{Var}} \colon \mathcal{SEM} \to \mathcal{SEM}_x$ from the category $\mathcal{SEM}$ of all predicates to the subcategory $\mathcal{SEM}_x$ of all those predicates that do not depend on $x$ with the semantic "injection" functor $J_x \colon \mathcal{SEM}_x \to \mathcal{SEM}$ whose maps are just identities, i.e., $J_x(Q) = Q$ and $J_x(f \colon P \to Q) = f \colon P \to Q$. We then have

$$\begin{gathered} J_x(\mathit{FORALL}_x(P)) \subseteq P \\ (J_x(Q) \subseteq P) \Rightarrow (Q \subseteq \mathit{FORALL}_x(P)) \end{gathered}$$

and dually

$$\begin{gathered} P \subseteq J_x(\mathit{EXISTS}_x(P)) \\ (P \subseteq J_x(Q)) \Rightarrow (\mathit{EXISTS}_x(P) \subseteq Q) \end{gathered}$$

Now, the first set of rules matches the assumptions of the second part of Proposition 7 for $F := J_x$ and $G := \mathit{FORALL}_x$ (considering that the subset relation $\subseteq$ denotes the existence of an arrow in categories $\mathcal{SEM}$, respectively $\mathcal{SEM}_x$); thus, we have $J_x \dashv \mathit{FORALL}_x$. Likewise, the second set of rules matches the assumptions of the first part of that proposition for $F := \mathit{EXISTS}_x$ and $G := J_x$; thus, we have $\mathit{EXISTS}_x \dashv J_x$. Summarizing, the universal functor $\mathit{FORALL}_x$ is the right adjoint of the injection functor $J_x$ while the existential functor $\mathit{EXISTS}_x$ is its left adjoint.

The above rules say that $\mathit{FORALL}_x(P)$ is the weakest predicate $Q$ that does not depend on $x$ for which $(J_x(Q) \subseteq Q)$ holds while $\mathit{EXISTS}_x(P)$ is the strongest predicate $Q$ that does not depend on $x$ for which $(Q \subseteq J_x(Q))$ holds. Since left and right adjoints are uniquely defined, these rules represent implicit but unique definitions of $\mathit{FORALL}_x(P)$ and $\mathit{EXISTS}_x(P)$ which can be explicitly written as follows:

$$\begin{aligned} \mathit{FORALL}_x(P) &:= \bigcup \{Q \in \mathit{Pred}_x \mid J_x(Q) \subseteq P\} \\ \mathit{EXISTS}_x(P) &:= \bigcap \{Q \in \mathit{Pred}_x \mid P \subseteq J_x(Q)\} \end{aligned}$$

From $J_x(Q) = Q$ and $Q \in \mathit{Pred}_x \Leftrightarrow Q \in \mathit{Pred} \wedge Q \perp\!\!\!\perp x$ (see Definition 5), this can also be written as follows:

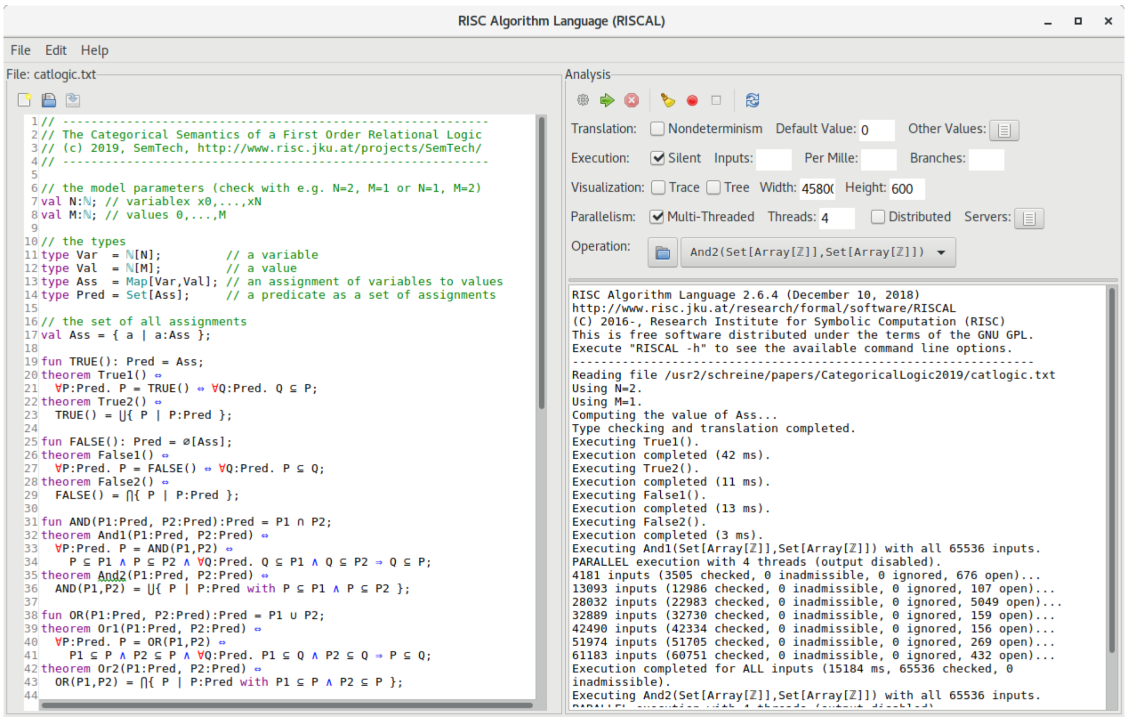
$$\begin{aligned} \mathit{FORALL}_x(P) &:= \bigcup \{Q \in \mathit{Pred} \mid Q \perp\!\!\!\perp x \wedge Q \subseteq P\} \\ \mathit{EXISTS}_x(P) &:= \bigcap \{Q \in \mathit{Pred} \mid Q \perp\!\!\!\perp x \wedge P \subseteq Q\} \end{aligned}$$

Thus, we have derived alternative characterizations $[\![ \forall x.\, F ]\!] = \mathit{FORALL}_x([\![ F ]\!])$ and $[\![ \exists x.\, F ]\!] = \mathit{EXISTS}_x([\![ F ]\!])$ that are both constructive and universal. This is exactly the characterization whose correctness we have proved in Proposition 6.

## 5. An Implementation of the Categorical Semantics

In this section, we describe how the constructions that we have theoretically modeled in Section 2 can be actually implemented. For this purpose, we use RISCAL (RISCAL is developed at JKU, Linz, Austria, https://www3.risc.jku.at/research/formal/software/RISCAL/, see [13]), the RISC Algorithm Language [13,17], a specification language, and an associated software system for modeling mathematical theories and algorithms in a specification language based on first-order logic and set theory. The language is based on a type system where all types have finite sizes (specified by the user);

this allows for fully automatically deciding formulas and verifying the correctness of algorithms for all possible inputs. To this end, the system translates every syntactic phrase into an executable form of its denotational semantics; the RISCAL model checker evaluates these semantics to determine the results of algorithms and the truth values of formulas such as the postconditions of algorithms. Since the domains of RISCAL models have (parameterized but) finite size, the validity of all theorems and the correctness of all algorithms can be fully automatically checked; the system has been mainly employed in educational scenarios [18,19]. Figure 2 gives a screenshot of the software with the RISCAL model that is going to be discussed below.



```
// ------------------------------------------------------------
// The Categorical Semantics of a First Order Relational Logic
// (c) 2019, SemTech, http://www.risc.jku.at/projects/SemTech/
// ------------------------------------------------------------

// the model parameters (check with e.g. N=2, M=1 or N=1, M=2)
val N:ℕ; // variablex x0,...,xN
val M:ℕ; // values 0,...,M

// the types
type Var  = ℕ[N];         // a variable
type Val  = ℕ[M];         // a value
type Ass  = Map[Var,Val]; // an assignment of variables to values
type Pred = Set[Ass];     // a predicate as a set of assignments

// the set of all assignments
val Ass = { a | a:Ass };

fun TRUE(): Pred = Ass;
theorem True1() ⇔
  ∀P:Pred. P = TRUE() ⇔ ∀Q:Pred. Q ⊆ P;
theorem True2() ⇔
  TRUE() = ⋃{ P | P:Pred };

fun FALSE(): Pred = ∅[Ass];
theorem False1() ⇔
  ∀P:Pred. P = FALSE() ⇔ ∀Q:Pred. P ⊆ Q;
theorem False2() ⇔
  FALSE() = ⋂{ P | P:Pred };

fun AND(P1:Pred, P2:Pred):Pred = P1 ∩ P2;
theorem And1(P1:Pred, P2:Pred) ⇔
  ∀P:Pred. P = AND(P1,P2) ⇔
    P ⊆ P1 ∧ P ⊆ P2 ∧ ∀Q:Pred. Q ⊆ P1 ∧ Q ⊆ P2 ⇒ Q ⊆ P;
theorem And2(P1:Pred, P2:Pred) ⇔
  AND(P1,P2) = ⋃{ P | P:Pred with P ⊆ P1 ∧ P ⊆ P2 };

fun OR(P1:Pred, P2:Pred):Pred = P1 ∪ P2;
theorem Or1(P1:Pred, P2:Pred) ⇔
  ∀P:Pred. P = OR(P1,P2) ⇔
    P1 ⊆ P ∧ P2 ⊆ P ∧ ∀Q:Pred. P1 ⊆ Q ∧ P2 ⊆ Q ⇒ P ⊆ Q;
theorem Or2(P1:Pred, P2:Pred) ⇔
  OR(P1,P2) = ⋂{ P | P:Pred with P1 ⊆ P ∧ P2 ⊆ P };

```

Translation: ☐ Nondeterminism Default Value: 0 Other Values:
Execution: ☑ Silent Inputs: Per Mille: Branches:
Visualization: ☐ Trace ☐ Tree Width: 4580( Height: 600
Parallelism: ☑ Multi-Threaded Threads: 4 ☐ Distributed Servers:
Operation: And2(Set[Array[ℤ]],Set[Array[ℤ]])

```
RISC Algorithm Language 2.6.4 (December 10, 2018)
http://www.risc.jku.at/research/formal/software/RISCAL
(C) 2016-, Research Institute for Symbolic Computation (RISC)
This is free software distributed under the terms of the GNU GPL.
Execute "RISCAL -h" to see the available command line options.
------------------------------------------------------------------
Reading file /usr2/schreine/papers/CategoricalLogic2019/catlogic.txt
Using N=2.
Using M=1.
Computing the value of Ass...
Type checking and translation completed.
Executing True1().
Execution completed (42 ms).
Executing True2().
Execution completed (11 ms).
Executing False1().
Execution completed (13 ms).
Executing False2().
Execution completed (3 ms).
Executing And1(Set[Array[ℤ]],Set[Array[ℤ]]) with all 65536 inputs.
PARALLEL execution with 4 threads (output disabled).
4181 inputs (3505 checked, 0 inadmissible, 0 ignored, 676 open)...
13093 inputs (12986 checked, 0 inadmissible, 0 ignored, 107 open)...
28032 inputs (22983 checked, 0 inadmissible, 0 ignored, 5049 open)...
32889 inputs (32730 checked, 0 inadmissible, 0 ignored, 159 open)...
42490 inputs (42334 checked, 0 inadmissible, 0 ignored, 156 open)...
51974 inputs (51705 checked, 0 inadmissible, 0 ignored, 269 open)...
61183 inputs (60751 checked, 0 inadmissible, 0 ignored, 432 open)...
Execution completed for ALL inputs (15184 ms, 65536 checked, 0
inadmissible).
Executing And2(Set[Array[ℤ]],Set[Array[ℤ]]) with all 65536 inputs.
```

**Figure 2.** The RISCAL Software.

Figures 3 and 4 list a RISCAL model of the categorical semantics over a domain of $N+1$ variables (identified with the natural numbers $0,\dots,N$) with $M+1$ values, for arbitrary model parameters $N, M \in \mathbb{N}_0$; all theorems over these domains are decidable and can be checked by RISCAL. The RISCAL definition of domains, functions, and predicates closely correspond to those given in this paper; in particular, we have a domain `Pred` of predicates (since the number of variables is finite, by definition all relations are predicates) and predicate functions `TRUE`, `FALSE`, `AND`, `OR`, `IMP`, `FORALL`, `EXISTS`. Different from the categorical formulation, `IMP` is a binary function, not a family of unary functions; likewise, `FORALL` and `EXISTS` are binary functions whose first argument is a variable. Furthermore, we introduce functions `NOT` and `EQUIV` for the semantics of negation and conjunction and show by theorems `Not` and `Equiv` that they can be reduced to the other functions.

```
// ------------------------------------------------------------
// The Categorical Semantics of a First Order Relational Logic
// (c) 2019, SemTech, http://www.risc.jku.at/projects/SemTech/
// ------------------------------------------------------------

// the model parameters (check with e.g. N=2, M=1 or N=1, M=2)
val N:ℕ; // variablex x0,...,xN
val M:ℕ; // values 0,...,M

// the types
type Var  = ℕ[N];          // a variable
type Val  = ℕ[M];          // a value
type Ass  = Map[Var,Val]; // an assignment of variables to values
type Pred = Set[Ass];      // a predicate as a set of assignments

// the set of all assignments
val Ass = { a | a:Ass };

fun TRUE(): Pred = Ass;
theorem True1() ⇔
  ∀P:Pred. P = TRUE() ⇔ ∀Q:Pred. Q ⊆ P;
theorem True2() ⇔
  TRUE() = ⋃{ P | P:Pred };

fun FALSE(): Pred = ∅[Ass];
theorem False1() ⇔
  ∀P:Pred. P = FALSE() ⇔ ∀Q:Pred. P ⊆ Q;
theorem False2() ⇔
  FALSE() = ⋂{ P | P:Pred };

fun AND(P1:Pred, P2:Pred):Pred = P1 ∩ P2;
theorem And1(P1:Pred, P2:Pred) ⇔
  ∀P:Pred. P = AND(P1,P2) ⇔
    P ⊆ P1 ∧ P ⊆ P2 ∧ ∀Q:Pred. Q ⊆ P1 ∧ Q ⊆ P2 ⇒ Q ⊆ P;
theorem And2(P1:Pred, P2:Pred) ⇔
  AND(P1,P2) = ⋃{ P | P:Pred with P ⊆ P1 ∧ P ⊆ P2 };

fun OR(P1:Pred, P2:Pred):Pred = P1 ∪ P2;
theorem Or1(P1:Pred, P2:Pred) ⇔
  ∀P:Pred. P = OR(P1,P2) ⇔
    P1 ⊆ P ∧ P2 ⊆ P ∧ ∀Q:Pred. P1 ⊆ Q ∧ P2 ⊆ Q ⇒ P ⊆ Q;
theorem Or2(P1:Pred, P2:Pred) ⇔
  OR(P1,P2) = ⋂{ P | P:Pred with P1 ⊆ P ∧ P2 ⊆ P };

fun IMP(P1:Pred, P2:Pred):Pred = (Ass\P1) ∪ P2;
theorem Imp1(P1:Pred, P2:Pred) ⇔
  ∀P:Pred. P = IMP(P1,P2) ⇔
    P ∩ P1 ⊆ P2 ∧ ∀Q:Pred. Q ∩ P1 ⊆ P2 ⇒ Q ⊆ P;
theorem Imp2(P1:Pred, P2:Pred) ⇔
  IMP(P1,P2) = ⋃{ P | P:Pred with P ∩ P1 ⊆ P2 };

fun NOT(P:Pred): Pred = Ass\P;
theorem Not(P:Pred) ⇔ NOT(P) = IMP(P,FALSE());

fun EQUIV(P1:Pred, P2:Pred): Pred =
  ((Ass\P1)∪P2) ∩ ((Ass\P2)∪P1);
theorem Equiv(P1:Pred, P2:Pred) ⇔
  EQUIV(P1,P2) = AND(IMP(P1,P2),IMP(P2,P1));
...
```

**Figure 3.** A RISCAL Model of the Categorical Semantics (Part 1).

```
...

pred independent(P:Pred, x:Var) ⇔
  ∀a:Ass, v1:Val, v2:Val.
    (a with [x] = v1) ∈ P ⇔ (a with [x] = v2) ∈ P;

fun FORALL(x:Var, P:Pred):Pred =
  { a | a:Ass with ∀v:Val. (a with [x] = v) ∈ P } ;
theorem Forall1(x:Var, P:Pred) ⇔
  ∀Q:Pred with independent(Q,x). Q = FORALL(x,P) ⇔
    Q ⊆ P ∧ ∀Q0:Pred with independent(Q0,x). Q0 ⊆ P ⇒ Q0 ⊆ Q;
theorem Forall2(x:Var, P:Pred) ⇔
  FORALL(x,P) = ⋃{ Q | Q:Pred with independent(Q,x) ∧ Q ⊆ P };

fun EXISTS(x:Var, P:Pred):Pred =
  { a | a:Ass with ∃v:Val. (a with [x] = v) ∈ P } ;
theorem Exists1(x:Var, P:Pred) ⇔
  ∀Q:Pred with independent(Q,x). Q = EXISTS(x,P) ⇔
    P ⊆ Q ∧ ∀Q0:Pred with independent(Q0,x). P ⊆ Q0 ⇒ Q ⊆ Q0;
theorem Exists2(x:Var, P:Pred) ⇔
  EXISTS(x,P) = ⋂{ Q | Q:Pred with independent(Q,x) ∧ P ⊆ Q };

// ------------------------------------------------------------
// end of file
// ------------------------------------------------------------
```

**Figure 4.** A RISCAL Model of the Categorical Semantics (Part 2).

All other logical operations are first defined in their usual set-theoretic form. Subsequently, we describe their categorical semantics by a pair of theorems: the first theorem claims that the set-theoretic semantics is equivalent to an implicit definition of the categorical semantics while the second theorem claims equivalence to the corresponding constructive definition. Choosing small parameter values $N = 2$ and $M = 1$ (i.e., relations with variables $x_0, x_1, x_2$ and values $0, 1$), RISCAL can easily check the validity of all claims, as demonstrated by the following output:

```
RISC Algorithm Language 2.6.4 (10 December 2018)
http://www.risc.jku.at/research/formal/software/RISCAL
(C) 2016-, Research Institute for Symbolic Computation (RISC)
This is free software distributed under the terms of the GNU GPL.
Execute "RISCAL -h" to see the available command line options.
-----------------------------------------------------------------
Reading file /usr2/schreine/papers/CategoricalLogic2019/catlogic.txt
Using N=2.
Using M=1.
Computing the value of Ass...
Computing the value of TRUE...
Computing the value of FALSE...
Type checking and translation completed.
Executing True1().
Execution completed (3 ms).
Executing True2().
Execution completed (1 ms).
Executing False1().
Execution completed (0 ms).
Executing False2().
Execution completed (1 ms).
Executing And1(Set[Array[ℤ]],Set[Array[ℤ]]) with all 65536 inputs.
PARALLEL execution with 4 threads (output disabled).
...
Execution completed for ALL inputs (18,373 ms, 65,536 checked, 0 inadmissible).
Executing And2(Set[Array[ℤ]],Set[Array[ℤ]]) with all 65,536 inputs.
PARALLEL execution with 4 threads (output disabled).
46273 inputs (36446 checked, 0 inadmissible, 0 ignored, 9827 open)...
Execution completed for ALL inputs (3576 ms, 65536 checked, 0 inadmissible).
Executing Or1(Set[Array[ℤ]],Set[Array[ℤ]]) with all 65536 inputs.
PARALLEL execution with 4 threads (output disabled).
...
Execution completed for ALL inputs (26,889 ms, 65,536 checked, 0 inadmissible).
Executing Or2(Set[Array[ℤ]],Set[Array[ℤ]]) with all 65,536 inputs.
```

```
PARALLEL execution with 4 threads (output disabled).
42,676 inputs (32,887 checked, 0 inadmissible, 0 ignored, 9789 open)...
Execution completed for ALL inputs (3907 ms, 65,536 checked, 0 inadmissible).
Executing Imp1(Set[Array[ℤ]],Set[Array[ℤ]]) with all 65,536 inputs.
PARALLEL execution with 4 threads (output disabled).
...
Execution completed for ALL inputs (48,592 ms, 65,536 checked, 0 inadmissible).
Executing Imp2(Set[Array[ℤ]],Set[Array[ℤ]]) with all 65,536 inputs.
PARALLEL execution with 4 threads (output disabled).
...
Execution completed for ALL inputs (9462 ms, 65,536 checked, 0 inadmissible).
Executing Not(Set[Array[ℤ]]) with all 256 inputs.
PARALLEL execution with 4 threads (output disabled).
Execution completed for ALL inputs (28 ms, 256 checked, 0 inadmissible).
Executing Equiv(Set[Array[ℤ]],Set[Array[ℤ]]) with all 65,536 inputs.
PARALLEL execution with 4 threads (output disabled).
Execution completed for ALL inputs (354 ms, 65,536 checked, 0 inadmissible).
Executing Forall1(ℤ,Set[Array[ℤ]]) with all 768 inputs.
PARALLEL execution with 4 threads (output disabled).
Execution completed for ALL inputs (1315 ms, 768 checked, 0 inadmissible).
Executing Forall2(ℤ,Set[Array[ℤ]]) with all 768 inputs.
PARALLEL execution with 4 threads (output disabled).
Execution completed for ALL inputs (512 ms, 768 checked, 0 inadmissible).
Executing Exists1(ℤ,Set[Array[ℤ]]) with all 768 inputs.
PARALLEL execution with 4 threads (output disabled).
Execution completed for ALL inputs (1299 ms, 768 checked, 0 inadmissible).
Executing Exists2(ℤ,Set[Array[ℤ]]) with all 768 inputs.
PARALLEL execution with 4 threads (output disabled).
Execution completed for ALL inputs (461 ms, 768 checked, 0 inadmissible).
```

These values are, however, the largest ones with which model checking is realistically feasible; choosing, for example, $N = 3$ and $M = 2$ gives for the checking theorem `And1` about $4 \times 10^9$ possible inputs whose checking on a single processor core would take RISCAL more than two decades.

## 6. Conclusions

In this paper, we developed a novel categorical interpretation of the semantics of a relational (term-less) variant of first-order logic [20] with the goal to aid the intuitive understanding of these formulas and to lay the seed of tools that illustrate the meaning of these formulas by the visualization of their semantics. The main advantage of this formulation is the (in comparison to other previous approaches) much more explicit illustration of the various logical constructions (connectives and quantifiers) as categorical notions; this may provide an alternative route to teaching semantics for first-order logic. Furthermore, since this categorical semantics is constructive, we could directly implement (and thus validate) it in the RISCAL software.

We hope that this paper, by its self-contained nature and by focusing on the core principles of categorical logic rather than attempting an exhaustive treatment, contributes to the more widespread dissemination of categorical ideas to students and researchers of logic, its applications, and its automation; in particular, it may provide an alternative view on the semantics of first-order logic by complementing the classical formulation and thus help to gain deeper insights.

It remains to be shown, however, whether and how this view can be indeed helpful and illuminating in educational scenarios, i.e., in courses on logic and its applications. In previous works [21–24], we have strived to improve the understanding of the formal semantics of programming languages by developing corresponding tools with appropriate visualization techniques, partially also based on categorical principles. Other work of ours [25] has extended this work towards the visualization of the semantics of first-order formulas by pruned evaluation trees, however, based on the classical formulation. Future work of us will investigate how the categorical principles outlined in this paper can be transferred to corresponding novel tools and visualization techniques for education in semantics and logic—for instance, in describing software component systems whose semantics are described both from the categorical and the logical side, or elaborating a precise logical definition of contracts that have to be satisfied for a successful composition of components.

**Author Contributions:** Conceptualization, W.S. (Wolfgang Schreiner), W.S. (William Steingartner) and V.N.; methodology, W.S. (Wolfgang Schreiner) and W.S. (William Steingartner); software, W.S. (Wolfgang Schreiner); validation, W.S. (Wolfgang Schreiner) and W.S. (William Steingartner); formal analysis, W.S. (Wolfgang Schreiner) and W.S. (William Steingartner); investigation, W.S. (Wolfgang Schreiner), W.S. (William Steingartner) and V.N.; resources, W.S. (Wolfgang Schreiner); data curation, W.S. (Wolfgang Schreiner); writing—original draft preparation, W.S. (Wolfgang Schreiner); writing—review and editing, W.S. (William Steingartner); visualization, W.S. (Wolfgang Schreiner); supervision, W.S. (Wolfgang Schreiner) and W.S. (William Steingartner); project administration, W.S. (Wolfgang Schreiner) and W.S. (William Steingartner); funding acquisition, W.S. (Wolfgang Schreiner) and W.S. (William Steingartner). All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the the project KEGA 011TUKE-4/2020: "A development of the new semantic technologies in educating of young IT experts", also in the frame of the initiative project "Semantic Modeling of Component-Based Program Systems" under the bilateral program "Aktion Österreich–Slowakei, Wissenschafts- und Erziehungskooperation" and by the Johannes Kepler University Linz, Linz Institute of Technology (LIT), Project LOGTECHEDU "Logic Technology for Computer Science Education".

**Conflicts of Interest:** The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

## References

1. Tarski, A. The Semantic Conception of Truth: In addition, the Foundations of Semantics. *Philos. Phenomenol. Res.* **1944**, *4*, 341–376. [CrossRef]
2. Schmidt, D.A. *Denotational Semantics—A Methodology for Language Development*; Allyn and Bacon: Boston, MA, USA, 1986.
3. Awodey, S. *Category Theory*, 2nd ed.; Oxford University Press: Okford, UK, 2010.
4. Barr, M.; Wells, C. *Category Theory for Computing Science*; Prentice-Hall, Inc., Division of Simon and Schuster One Lake Street: Upper Saddle River, NJ, USA, 1990.
5. Brandenburg, M. *Einführung in Die Kategorientheorie*; Springer Spektrum: Berlin/Heidelberg, Germany, 2017. (In German)
6. Pierce, B.C. *Basic Category for Computer Scientists*; MIT Press: Cambridge, MA, USA, 1991.
7. Spiwak, D.I. *Category Theory for the Sciences*; MIT Press: Cambridge, MA, USA, 2014.
8. Lawvere, F.W. Adjointness in Foundations. *Dialectica* **1969**, *23*, 281–296. Available online: http://www.tac.mta.ca/tac/reprints/articles/16/tr16.pdf (accessed on 9 March 2020). [CrossRef]
9. Jacobs, B. *Categorical Logic and Type Theory*; Studies in Logic and the Foundations of Mathematics; Elsevier: Amsterdam, The Netherlands, 1999, Volume 141.
10. Abramsky, S. Logic and Categories As Tools For Building Theories. *J. Indian Counc. Philos. Res. Issue Log. Philos. Today* **2010**, *27*, 277–304.
11. Poigné, A. Category Theory and Logic. In *Proceedings of the Category Theory and Computer Programming: Tutorial and Workshop, Guildford, UK, 16–20 September 1985*; Lecture Notes in Computer Science; Springer: Berlin, Germany; Volume 240, pp. 103–142. [CrossRef]
12. Abramsky, S.; Tzevelekos, N. Introduction to Categories and Categorical Logic. In *New Structures for Physics*; Lecture Notes in Physics; Springer: Berlin, Germany, 2010; Volume 813, pp. 3–94. [CrossRef]
13. RISCAL. The RISC Algorithm Language (RISCAL). Available online: https://www3.risc.jku.at/research/formal/software/RISCAL/ (accessed on 9 July 2020).
14. Schreiner, W.; Reichl, F.X. Mathematical Model Checking Based on Semantics and SMT. *Trans. Internet Res.* **2020**, *16*, 4–13.
15. Semtech. Semantic Technologies for Computer Science Education. Available online: https://www3.risc.jku.at/projects/SemTech/ (accessed on 15 July 2020).
16. Nielson, H.R.; Nielson, F. *Semantics with Applications: An Appetizer*; Undergraduate Topics in Computer Science; Springer: London, UK, 2007.
17. Schreiner, W. *The RISC Algorithm Language (RISCAL)—Tutorial and Reference Manual (Version 1.0)*; Technical Report; RISC, Johannes Kepler University: Linz, Austria, 2017.

18. Schreiner, W. Validating Mathematical Theories and Algorithms with RISCAL. In *Proceedings of the 11th Conference on Intelligent Computer Mathematics (CICM 2018), Hagenberg, Austria, 13–17 August 2018*; Rabe, F., Farmer, W., Passmore, G., Youssef, A., Eds.; Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence; Springer: Berlin, Germany, 2018; Volume 11006, pp. 248–254. [CrossRef]
19. Schreiner, W.; Brunhuemer, A.; Fürst, C. Teaching the Formalization of Mathematical Theories and Algorithms via the Automatic Checking of Finite Models. In Proceedings of the Post-Proceedings ThEdu'17, Theorem Proving Components for Educational Software, Gothenburg, Sweden, 6 August 2017.
20. Schreiner, W.; Novitzká, V.; Steingartner, W. *A Categorical Semantics of Relational First-Order Logic*; Technical Report; RISC, Johannes Kepler University: Linz, Austria, 2019.
21. Schreiner, W.; Steingartner, W. *Visualizing Execution Traces in RISCAL*; Technical Report; RISC, Johannes Kepler University: Linz, Austria, 2018.
22. Steingartner, W.; Eldojali, M.A.M.; Radaković, D.; Dostál, J. Software support for course in Semantics of programming languages. In Proceedings of the IEEE 14th International Scientific Conference on Informatics, Poprad, Slovakia, 14–16 November 2017; pp. 359–364.
23. Steingartner, W.; Novitzká, V. Categorical Semantics of Programming Langages. In *Selected Topics in Contemporary Mathematical Modeling*; Monographs; Czestochowa University of Technology, Czestochowa, Poland: 2017; Volume 331, Chapter 11, pp. 167–192.
24. Steingartner, W.; Novitzká, V. Learning tools in course on semantics of programming languages. In Proceedings of the MMFT 2017—Mathematical Modelling in Physics and Engineering, Poraj, Poland, 18–21 September 2017; pp. 137–142.
25. Schreiner, W.; Steingartner, W. *Visualizing Logic Formula Evaluation in RISCAL*; Technical Report; RISC, Johannes Kepler University: Linz, Austria, 2018.