Technical Report: Initial Algebra Semantics in Matching Logic

Xiaohong Chen[1], Dorel Lucanu[2], and Grigore Roşu[1]

[1]University of Illinois at Urbana-Champaign, Champaign, USA
[2]Alexandru Ioan Cuza University, Iaşi, Romania
`xc3@illinois`, `dlucanu@info.uaic.ro`, `grosu@illinois.edu`

July 24, 2020

Abstract

Matching logic is a unifying foundational logic for defining formal programming language semantics, which adopts a minimalist design with few primitive constructs that are enough to express all properties within a variety of logical systems, including FOL, separation logic, (dependent) type systems, modal μ-logic, and more. In this paper, we consider *initial algebra semantics* and show how to capture it by matching logic specifications. Formally, given an algebraic specification E that defines a set of sorts (of data) and a set of operations whose behaviors are defined by a set of equational axioms, we define a corresponding matching logic specification, denoted $\mathsf{INITIALALGEBRA}(E)$, whose models are exactly the initial algebras of E. Thus, we reduce initial E-algebra semantics to the matching logic specifications $\mathsf{INITIALALGEBRA}(E)$, and reduce extrinsic initial E-algebra reasoning, which includes inductive reasoning, to generic, intrinsic matching logic reasoning.

1 Introduction

Initial algebra semantics is a main approach to formal programming language semantics based on algebraic specifications and their initial models. It originated in the 1970s, when algebraic techniques started to be applied to specify basic data types such as lists, trees, stacks, etc. The original paper on initial algebra semantics [Goguen et al., 1977] reviewed various existing algebraic specification techniques and showed that they were all *initial models*, making the concept of *initiality* explicit for the first time in formal language semantics. Since 1977, initial algebra semantics has gathered much research interest and become a well-established field, leading to a profound study on its foundations as well as applications, tools, libraries, packages, provers [Goguen et al., 2000; Clavel et al., 2020; Diaconescu and Futatsugi, 1998; Pitts, 2019; van den Brand et al., 2001; Astesiano et al., 2002].

The key idea of initial algebra semantics is as follows. Let E be an algebraic specification that defines the sorts of data and the operations on the data, whose behaviors are axiomatized by equations. In the class C of all algebras satisfying E, an algebra $I \in C$ is *initial* iff for any $A \in C$ there is a unique morphism $h_A : I \to A$. According to this view, program syntax *is* an initial algebra, and C includes all possible implementations, or semantic models, of E. A formal semantics is then a way to associate the syntax with the *intended* semantic model $A \in C$, where the unique morphism h_A is the semantic function mapping syntax to semantics. As an example, the Scott-Strachey denotational semantics [Scott, 1982] uses complete partial orders as the intended semantic models, and inductively defines denotations for each syntactic constructs to be the unique morphisms.

Initiality has a close relationship with *induction*. Since program syntax is often defined inductively, the initial algebra I enjoys the *principle of induction*, which can then be mapped to the semantic models through the unique morphisms. Note that this use of the initiality often occurs without being noticed; e.g. when we apply "structural induction on program syntax" to prove "properties about the semantic models", we are mapping the inductive reasoning on the initial algebra (program syntax) to its semantics, via the unique morphism. We explain this in detail in Section 9.

Contribution

We show how to internalize initial algebra semantics within *matching logic* [Roşu, 2017; Chen and Roşu, 2019a, 2020a], the logical foundation of the language semantics framework $\mathbb{K}$ (`http://kframework.org`), which was used to define complete formal semantics to several real-world languages: C [Hathhorn et al., 2015], Java [Bogdănaş and Roşu, 2015], JavaScript [Park et al., 2015], Ethereum VM [Hildenbrandt et al., 2018], x86 [Dasgupta et al., 2019], etc. The gist of matching logic is that it allows us to specify and reason about program configurations compactly and modularly, using a formalism that keeps and respects the original syntactic/semantic structures (programs, continuations, heaps, stacks, etc) as are, without losing them in translations. Matching logic is *minimal* and *expressive.* Its formulas, called *patterns*, are built from only these syntactic constructs:

$$\varphi ::= x \mid X \mid \sigma \mid \varphi_1\ \varphi_2 \mid \bot \mid \varphi_1 \to \varphi_2 \mid \exists x.\, \varphi \mid \mu X.\, \varphi$$

where x ranges over elements; X ranges over sets; σ is a symbol that can represent functions, predicates, modal operators, etc., and can be *applied* to other patterns like in $(\sigma\ \varphi)$; $\bot$ and $\varphi_1 \to \varphi_2$ build propositional constraints; $\exists$ builds quantification; and μ builds least fixpoints (induction). The entire metatheory of matching logic, including proof theory and model theory, can be presented in one page (Fig. 17), and a proof checker can be implemented in fewer than 250 LOC [K Team, 2020]. Yet, matching logic is very expressive and captures many logical systems *axiomatically* as specifications, including FOL, separation logic, λ-calculus, (dependent) type systems, Hoare logic, rewriting logic, reachability logic, temporal logics, and modal μ-logic [Roşu, 2017; Chen and Roşu, 2019a, 2020a]. Unfortunately, matching logic is only partly supported by $\mathbb{K}$, as a foundation for its decision procedures, interactive theorem proving, and proof search heuristics. $\mathbb{K}$'s support for initial algebra semantics is limited and hidden under the hood; users cannot make explicit use of matching logic's fixpoint patterns yet to carry out inductive reasoning. We are actively working with the $\mathbb{K}$ team to adopt the results in this paper and incorporate them in their interactive theorem prover.

Organization

We review related work on initial algebra in Section 2 and basic concepts in Section 3. Then we propose a new variant of matching logic that has direct support for induction in Section 4 and use it to methodologically capture sorts in Section 5. Then, in Section 6, we define the matching logic specification $\mathsf{EQSPEC}(E)$ that captures all the algebras that satisfy a given algebraic equational specification E; in Sections 7-8, we define the matching logic specification $\mathsf{INITIALALGEBRA}(E)$ that captures precisely the initial algebras of E; term algebras are considered as a special case in Section 7; in Section 9, we show how $\mathsf{INITIALALGEBRA}(E)$ supports inductive reasoning in initial algebras. We discuss extensions of initial algebra semantics in Section 10 and conclude in Section 11.

The appendix contains all proof details.

2 Related Work

Related work on algebraic specification is vast. [Kutzler and Lichtenberger, 1983] gives an early bibliography of 870 papers that are about "algebraically specified abstract data types". Here, we focus on (1) the applications of initial algebra semantics to programming languages and (2) the related work that connect initial algebra semantics with other logical systems and/or frameworks.

The practical use of initial algebra semantics in programming started with the OBJ system [Goguen et al., 2000] based on order-sorted equational logic and parameterized programming. Its successors include Maude [Clavel et al., 2020] that is based on membership equational logic [Meseguer, 1997] and rewriting logic [Meseguer, 1992], and CafeOBJ [Diaconescu and Futatsugi, 1998] based on hidden algebra [Goguen and Malcolm, 2000]. Other systems include ASF+SDF [van den Brand et al., 2001] that uses initial semantics for its modules, and CASL [Astesiano et al., 2002] that allows to declaratively specify the initiality of the intended models. Initial algebra semantics also has applications in other aspects of programming. [Fiore et al., 1999] uses initial algebra semantics to define syntax with variable binding. [Dybjer, 1997] characterizes W-types as initial algebras of polynomial endofunctors, and the result is lifted to higher inductive types [Sojakova, 2015]. [Rutten and Turi, 1993] studies the duality between initial algebra semantics and final coalgebra semantics.

Many implementations of algebraic approaches are based on *type theory* using proof assistants like Coq [Coq Team, 2020] and Agda [Norell, 2009]. [Capretta, 1999] may be the first Coq implementation of the fundamentals such as signatures, algebras, quotient algebras, and term algebras, where sorts are defined as *setoids* (i.e., sets paired with an equivalence relation) and operations as constructors of the inductive types. The same idea is used by several frameworks of constructive algebraic hierarchy, consisting of implementations of various concrete algebras such as groups, rings, and fields [Geuvers et al., 2002; Spitters and van der Weegen, 2011; Garillot et al., 2009]. More recent work are based on homotopy type theory and formalize homotopy-initial algebras [Sojakova, 2015; Awodey et al., 2012, 2017; Kaposi et al., 2019; Fiore et al., 2020]. [Gunther et al., 2018] is a recent Agda implementation of algebraic specifications and initial algebras.

In general, modern type systems have more powerful features than initial algebra semantics, and it is not surprising that initiality can be defined by inductive types and their constructors, whose logical foundation, often a variant of (inductive) type theory, has complex and powerful proof rules (e.g., [Coq Team, 2020]) and native support for induction, fixpoints, and higher-order reasoning. The other direction is to give type systems an initial algebra semantics. [Coquand and Paulin, 1990] gives a set-theoretic semantics to inductive types using initial algebras. [Abel et al., 2008] gives an algebraic presentation of the Martin-Löf's intuitionistic type theory and uses initiality to formulate the correctness of the type-checking algorithm. [Johann and Ghani, 2007] suggests that initial algebra is sufficient as a semantics for functional programming languages with nested types.

Compared to powerful type systems and even initial algebra semantics, matching logic has a relatively small and straighforward proof/model theory (see Fig. 17). [Chen and Roşu, 2020a] investigates type systems in matching logic, but inductive reasoning is left as future work. This paper systematically studies induction via initial models; the resulting specification of initial algebra, which also fits in one page (Fig. 18), captures both the models and the formal inductive reasoning.

3 Initial Algebra Semantics

Here we review the main concepts, definitions, and results about initial algebra semantics. In this paper we follow the standard many-sorted approach in [Goguen and Meseguer, 1985; Meseguer and Goguen, 1985], which we recall in detail in Sections 3.1-3.4 for self-containedness and notation. The reader interested in variants, extensions, more insights and the history of initial algebra semantics, is referred to [Ehrig and Mahr, 1985; Manca and Salibra, 1992; Goguen et al., 1977; Goguen and Meseguer, 1992; Sannella and Tarlecki, 2012]; we include a brief discussion in Section 3.5.

3.1 Signatures, Algebras, and Terms

Definition 3.1. A *(many-sorted) signature* (S, F), often abbreviated F, consists of:

1. a finite set S of *sorts* denoted $s, s_1, s_2, \ldots$; and

2. a finite $(S^* \times S)$-indexed set $F = \{F_{s_1 \ldots s_n, s}\}_{s_1,\ldots,s_n,s \in S}$ of *operation symbols* or just *operations.*

For each $f \in F_{s_1 \dots s_n, s}$, we call $s_1, \dots, s_n$ the argument sorts and s the return sort, where n is the number of arguments that f takes. When $n = 0$, we write $f \in F_{\epsilon, s}$ and call it a *constant operation.*

Remark 3.2. The requirement that S and F are finite is for presentation convenience in this section. We discuss generalizations where S and F are allowed to be infinite in Section 10.

Definition 3.3. Given a signature (S, F), an *(S, F)-algebra* A, or simply an F-algebra, consists of:

1. a (possibly empty) *carrier set* A_s for each sort $s \in S$; and
2. an *operation interpretation* $A_f \colon A_{s_1} \times \cdots \times A_{s_n} \to A_s$ for each operation $f \in F_{s_1 \dots s_n, s}$.

Definition 3.4. Let (S, F) be a signature. An *S-sorted variable set* V is a set of variables denoted $x, y, \dots$ where each variable $x \in V$ is associated with its sort denoted $\mathsf{sort}(x) \in S$. The set $T_{F,s}(V)$ of *terms of sort s with variables from V* is inductively defined by the following grammar:

$$\begin{aligned} t_s &::= x && \text{where } x \in V \text{ and } \mathsf{sort}(x) = s \\ &\mid f(t_{s_1}, \dots, t_{s_n}) && \text{where } f \in F_{s_1 \dots s_n, s} \text{ and } t_{s_1} \in T_{F,s_1}(V), \dots, t_{s_n} \in T_{F,s_n}(V) \end{aligned}$$

We define $T_F(V) = \bigcup_{s \in S} T_{F,s}(V)$ as the set of all terms with variables from V. The set $T_{F,s} = T_{F,s}(\emptyset)$ consists of the *ground terms of sort s*. We define $T_F = \bigcup_{s \in S} T_{F,s}$ to be the set of all ground terms.

Definition 3.5. Let (S, F) be a signature, A be an (S, F)-algebra, and V be an S-sorted variable set. An *A-valuation* $\varrho \colon V \to A$ is a function such that $\varrho(x) \in A_{\mathsf{sort}(x)}$ for all $x \in V$. It yields a *term interpretation* $\bar{\varrho} \colon T_F(V) \to A$ as expected: (1) $\bar{\varrho}(x) = \varrho(x)$ for all $x \in V$; and (2) $\bar{\varrho}(f(t_{s_1}, \dots, t_{s_n})) = A_f\,(\bar{\varrho}(t_{s_1}), \dots, \bar{\varrho}(t_{s_n}))$ for all $f(t_{s_1}, \dots, t_{s_n}) \in T_F(V)$.

Remark 3.6. When $V = \emptyset$, the unique "empty valuation" is denoted $\emptyset \colon \emptyset \to A$. It yields the term interpretation $\bar{\emptyset} \colon T_F \to A$ for all ground terms. For clarity, we define $\mathsf{eval}_A(t) = \bar{\emptyset}(t)$ for all ground terms $t \in T_F$ and abbreviate it as $\mathsf{eval}(t)$ when A is understood.

3.2 Equational Specifications and Deduction

Definition 3.7. Given a signature (S, F), an *(F-)equation* has the form $\forall V.\, t = t'$ where V is a finite variable set and $t, t' \in T_{F,s}(V)$ are two terms of same sort s. We use $e, e_1, e_2, \dots$ to range over equations. When $V = \emptyset$, we call $\forall \emptyset.\, t = t'$ a *ground equation*, where $t, t' \in T_{F,s}$ are ground terms.

Definition 3.8. Given signature (S, F), an (S, F)-algebra A *validates* an equation $\forall V.\, t = t'$, or that the equation *holds* in A, written $A \vDash_{\mathsf{Alg}} \forall V.\, t = t'$, if $\bar{\varrho}(t) = \bar{\varrho}(t')$ for all valuations $\varrho \colon V \to A$.

Definition 3.9. An *equational specification* (S, F, E) consists of a signature (S, F) and a finite set E of F-equations. An *(S, F, E)-algebra*, or simply an (F, E)-algebra or just E-algebra, is an F-algebra A that validates all equations in E, written $A \vDash_{\mathsf{Alg}} E$. We define $E \vDash_{\mathsf{Alg}} e$ to mean that $A \vDash_{\mathsf{Alg}} e$ for all E-algebras A. We often abbreviate the equational specification (S, F, E) as (F, E) or simply E.

There are many (equivalent) equational proof systems in the literature; the following is standard:

Definition 3.10. For an equational specification E, we define *equational deduction* $E \vdash_{\mathsf{Alg}} e$ as:

1. axiom: if $(\forall V.\, t = t') \in E$ then $E \vdash_{\mathsf{Alg}} \forall V.\, t = t'$;
2. reflexivity: $E \vdash_{\mathsf{Alg}} \forall V.\, t = t$;
3. symmetry: if $E \vdash_{\mathsf{Alg}} \forall V.\, t = t'$ then $E \vdash_{\mathsf{Alg}} \forall V.\, t' = t$;
4. transitivity: if $E \vdash_{\mathsf{Alg}} \forall V.\, t = t'$ and $E \vdash_{\mathsf{Alg}} \forall V.\, t' = t''$ then $E \vdash_{\mathsf{Alg}} \forall V.\, t = t''$;
5. congruence: if $E \vdash_{\mathsf{Alg}} \forall V.\, t_i = t'_i$ for $1 \le i \le n$ then $E \vdash_{\mathsf{Alg}} \forall V.\, f(t_1, \dots, t_n) = f(t'_1, \dots, t'_n)$;
6. substitution: if $E \vdash_{\mathsf{Alg}} \forall V.\, t = t'$ and $\theta \colon V \to T_F(U)$ then $E \vdash_{\mathsf{Alg}} \forall U.\, \theta(t) = \theta(t')$.

The following theorem states that equational deduction is *sound* and *complete*.

Theorem 3.11. *For any equational specification E and equation e, $E \vdash_{\mathsf{Alg}} e$ if and only if $E \vDash_{\mathsf{Alg}} e$.*

3.3 Congruences and Quotient Algebras

Definition 3.12. Given a signature (S, F) and an F-algebra A, a *congruence* R on A is an S-indexed family of equivalence relations $R_s \subseteq A_s \times A_s$ for each sort $s \in S$, such that if $(a_i, b_i) \in R_{s_i}$ for $1 \leq i \leq n$ then $(A_f(a_1, \ldots, a_n), A_f(b_1, \ldots, b_n)) \in R_s$, for all $f \in F_{s_1 \ldots s_n, s}$ and $a_i, b_i \in A_{s_i}$, $1 \leq i \leq n$.

Definition 3.13. Given a signature (S, F), an F-algebra A, and a congruence R on A, we define the *R-quotient algebra $A_{/R}$* as the F-algebra that consists of:

1. the carrier set $A_{/R,s} = \{[a]_R \mid a \in A_s\}$ for each $s \in S$, where $[a]_R = \{b \in A_s \mid (a, b) \in R_s\}$ is the R-equivalence class of $a \in A_s$;
2. the operator interpretation $A_{/R,f} \colon A_{/R,s_1} \times \cdots \times A_{/R,s_n} \to A_{/R,s}$ for each $f \in F_{s_1 \ldots s_n, s}$, defined as $A_{/R,f}([a_1]_R, \ldots, [a_n]_R) = [A_f(a_1, \ldots, a_n)]_R$ for all $[a_i]_R \in A_{/R,s_i}$, $1 \leq i \leq n$.

Indeed, the operator interpretation $A_{/R,f}$ is well-defined because R is a congruence.

3.4 Term Algebras, Quotient Term Algebras, and Initiality

Definition 3.14. Given a signature (S, F), the *(S, F)-term algebra*, simply the F-term algebra or just term algebra, is the F-algebra that consists of:

1. the carrier set $T_s = T_{F,s}$ for $s \in S$, where $T_{F,s}$ is the set of all ground terms of sort s; and
2. the operator interpretation $T_f \colon T_{F,s_1} \times \cdots \times T_{F,s_n} \to T_{F,s}$ for each $f \in F_{s_1 \ldots s_n, s}$, defined as $T_f(t_{s_1}, \ldots, t_{s_n}) = f(t_{s_1}, \ldots, t_{s_n})$ for all $t_{s_i} \in T_{F,s_i}$, $1 \leq i \leq n$.

For notational convenience, we use T_F to denote the F-term algebra.

Equational deduction yields a congruence relation on the term algebra:

Proposition 3.15. *Given an equational specification (S, F, E), let $\simeq_E$ be the S-indexed family of relations $\simeq_{E,s} \subseteq T_{F,s} \times T_{F,s}$ for each sort $s \in S$, defined as $t \simeq_{E,s} t$ iff $E \vdash_{\mathsf{Alg}} \forall \emptyset .\, t = t'$ for all $t, t' \in T_{F,s}$. Then, $\simeq_E$ is a congruence on the term algebra T_F.*

Remark 3.16. We abbreviate $[t]_{\simeq_E}$ as $[t]_E$, or just $[t]$ when E is understood. We also abbreviate $t \simeq_{E,s} t'$ as $t \simeq_E t'$ or simply $t \simeq t'$. We write $T_{F/E,s} = \{[t]_E \mid t \in T_{F,s}\}$ as the set of $\simeq_E$-equivalence classes of terms of sort s and $T_{F/E} = \bigcup_{s \in S} T_{F/E,s}$ as the set of all $\simeq_E$-equivalence classes.

A *quotient term algebra* is then the $\simeq_E$-quotient of the term algebra. Formally,

Definition 3.17. Given an equational specification (S, F, E), the *(S, F, E)-quotient term algebra*, simply the (F, E)-quotient term algebra or just E-quotient term algebra, is the $\simeq_E$-quotient of the F-term algebra T_F, which for notational simplicity we denote as $T_{F/E}$. Specifically,

1. $T_{F/E,s}$ is the carrier set for each $s \in S$; and
2. $T_{F/E,f} \colon T_{F/E,s_1} \times \cdots \times T_{F/E,s_n} \to T_{F/E,s}$ is the operation interpretation for each $f \in F_{s_1 \ldots s_n, s}$, defined as $T_{F/E,f}([t_{s_1}], \ldots, [t_{s_n}]) = [f(t_{s_1}, \ldots, t_{s_n})]$ for all $[t_{s_1}] \in T_{F/E,s_1}, \ldots, [t_{s_n}] \in T_{F/E,s_n}$.

Term algebras and quotient term algebras are the canonical, concrete examples of *initial algebras.* In this paper we work with the most standard definition of initial algebra, which is that of an *initial object* in the category of algebras. We first recall algebra morphisms:

Definition 3.18. Let A and B be algebras of the same signature (S, F). An *(algebra) morphism* $h \colon A \to B$ is a function such that $h(A_f(a_1, \ldots, a_n)) = B_f(h(a_1), \ldots, h(a_n))$ for all $f \in F_{s_1 \ldots s_n, s}$ and $a_i \in A_{s_i}$, $1 \leq i \leq n$. In addition, if the inverse $h^{-1} \colon B \to A$ of h exists and is also a morphism, then h is an *isomorphism* and A and B are *isomorphic.* Isomorphic algebras are regarded as the same.

Definition 3.19. Let (S, F, E) be an equational specification. An *initial* (S, F, E)-algebra I, simply an initial (F, E)-algebra or just initial E-algebra, is an E-algebra such that for every E-algebra A, there exists a unique morphism $h_A : I \to A$. When $E = \emptyset$, we abbreviate initial $(S, F, \emptyset)$-algebras as the initial (S, F)-algebras or simply initial F-algebras.

Theorem 3.20. *The initial (F, E)-algebra exists and is unique up to isomorphism. In particular, the term algebra T_F is the initial F-algebra and the quotient term algebra $T_{F/E}$ is the initial (F, E)-algebra.*

Remark 3.21. Initial E-algebras are initial objects in the category of E-algebras.

As a specification, (F, E) states the existence of some data, operations, and equational properties. The initial (F, E)-algebra $T_{F/E}$ is a "minimal" realization of the specification (F, E), in the sense that all its elements are representable by F-terms and it satisfies no other equations except those derived from E. In other words, it satisfies the famous "no junk, no confusion" slogan, firstly proposed in [Burstall and Goguen, 1982]. In the following, we formally define the no-junk and no-confusion properties, and recall that an algebra satisfies no-junk and no-confusion iff it is the initial algebra.

Theorem 3.22. *Let (S, F, E) be an equational specification. For any E-algebra A, we say that:*

1. *A satisfies* no-confusion, *if for any ground equation e, we have $A \vDash_{\mathsf{Alg}} e$ implies $E \vdash_{\mathsf{Alg}} e$.*
2. *A satisfies* no-junk, *if for any element $a \in A$ there is a ground term t_a such that $\mathsf{eval}_A(t_a) = a$.*

A satisfies no-confusion and no-junk if and only if A is the initial (F, E)-algebra.

No-confusion states that A does not equate ground terms unless provably equal. In other words, if ground terms t and t' are interpreted the same in A, then $E \vdash_{\mathsf{Alg}} \forall\emptyset.\, t = t'$, which implies that $E \vDash_{\mathsf{Alg}} \forall\emptyset.\, t = t'$ (Theorem 3.11), that is, t and t' are interpreted the same in *all* E-algebras. No-junk states that A does not include elements that are not representable (or reachable) by ground terms.

Theorem 3.22 states that "no junk, no confusion" is an equivalent characterization of initial algebras. Compared with Definition 3.19, Theorem 3.22 is more convenient when we want to show that a given algebra is an initial algebra. Our matching logic specification of initial algebras was inspired by "no-junk and no-confusion", and in Sections 7 and 8 we will use Theorem 3.22 to show that the algebras contained within the matching logic models are indeed initial algebras.

3.5 Applications and Extensions

Initiality has been considered in various contexts, including many-sorted FOL [Wang, 1952; Enderton, 1972], order-sorted algebras [Goguen and Meseguer, 1992; Poigné, 1990], membership equational logic [Meseguer, 1997], parametric specifications [Bergstra and Klop, 1983; Ehrig et al., 1984], functor-based approaches [Rutten and Turi, 1993; Fiore and Hur, 2009], etc. The palette of initial semantics applications is also large, including abstract data types [Goguen et al., 1977; Guttag and Horning, 1978; Ehrig and Mahr, 1985], formal semantics [Guttag et al., 1985; Goguen and Malcolm, 1996], abstract syntax [Fiore et al., 1999], and inductive/dependent type systems used by proof assistants such as Agda [Norell, 2007], Coq [Coq Team, 2020], and Lean [de Moura et al., 2015].

Our matching logic approach to initial algebra semantics is not limited to the standard variant. In Section 10, we discuss three extensions that are known to be practical but also challenging. The first is *parametric* specifications that support sorts like $PList\langle s\rangle$, parametric in a sort s; (parametric) operations and axioms are defined once, and then instantiated by different sorts to avoid duplication. The second is *order-sorted* specifications that add a subsorting relation between sorts, e.g. $Nat \le Int$, to mean the inclusion between the corresponding data types in models. The third is *simultaneous inductive-recursive* definitions that support mutually recursive definitions of sets and operations.

4 A New Variant of Matching Logic

Matching logic was originally proposed in [Roşu and Schulte, 2009] as a means to specify and reason about programs compactly and modularly, using a formalism that keeps and respects the semantic structure and does not lose them through encodings or translations. The key concept in matching logic is that of *patterns*, which are used to specify the program configurations that *match* them. Since 2009, matching logic has been developed into a unifying logic for programming language semantics. On foundations, [Roşu, 2017; Chen and Roşu, 2019a, 2020a] show that matching logic captures various popular logical systems used in defining formal language semantics, such as FOL, Hoare logic [Hoare, 1969], separation logic [Reynolds, 2002], modal logics [Hughes and Cresswell, 1968], temporal logics [Prior, 1955], λ-calculus [Church, 1941], type systems [Martin-Löf, 1975], and so on, including their variants and extensions. On implementations, matching logic has been adopted as the logical foundation of the $\mathbb{K}$ formal semantic framework (`http://kframework.org`). The complete, executable formal semantics of many real-world languages, including C [Hathhorn et al., 2015], Java [Bogdănaş and Roşu, 2015], JavaScript [Park et al., 2015], Ethereum VM [Hildenbrandt et al., 2018], and x86 [Dasgupta et al., 2019], have been formalized in $\mathbb{K}$, which means that the formal semantics of these languages become matching logic specifications, and their formal analysis tools have been automatically generated by $\mathbb{K}$ from their formal semantics.

There are several variants of matching logic. Most of them have a many-sorted flavor where a sort set S is given and fixed by a (matching logic) signature. While the many-sorted setting is convenient for capturing models and structures that are also many-sorted, [Chen and Roşu, 2020a] pointed out that it actually becomes an obstacle in defining more complex sort/type structures, including those mentioned in Section 3.5. Therefore, they proposed a new variant as an alternative, called the *functional variant* of matching logic, where the many-sorted infrastructure is dropped and sorts are instead defined axiomatically. The authors then showed that the original many-sorted setting can be easily restored in the functional variant by appropriate axiomatization.

In this paper, we are interested in finding the most basic and simplest logic that can capture initial algebra semantics, and thus we build upon the functional variant of matching logic proposed in [Chen and Roşu, 2020a]. However, that variant has no support for induction or recursion or fixpoints, which are crucial to capture initiality. Therefore, our first contribution is an extension with *fixpoint patterns*. To avoid inventing new terminology, we still call our extension *matching logic*. We will thoroughly develop its metatheory. In particular, we define the semantics of fixpoint patterns by the Knaster-Tarski fixpoint theorem [Tarski, 1955] and propose a set of proof rules for fixpoint reasoning. In Sections 4.1-4.2, we define the syntax and semantics of matching logic, and in Section 4.3, we define matching logic specifications. In Section 4.4, we define the proof system.

4.1 Matching Logic Syntax

Throughout the paper, we fix two disjoint, unsorted variable sets EV and SV, where EV includes *element variables* denoted $x, y, \dots$ and SV includes *set variables* denoted $X, Y, \dots$.

Definition 4.1. Let Σ be an at most countable set of *(constant) symbols* called a *matching logic signature*. Symbols are denoted σ, σ', σ_1, σ_2, ... The set $\textsc{Pattern}_\Sigma$ of Σ-*patterns*, simply called patterns, is inductively defined by the following grammar, where $x \in EV$, $X \in SV$, and $\sigma \in \Sigma$:

$$\varphi ::= x \mid X \mid \sigma \mid \varphi_1\ \varphi_2 \mid \bot \mid \varphi_1 \to \varphi_2 \mid \exists x.\, \varphi \mid \mu X.\, \varphi \quad \text{if } \varphi \text{ is positive in } X \tag{1}$$

A pattern φ is positive in X iff X does not occur in an odd number of times of the left-hand sides of implications $\varphi_1 \to \varphi_2$. We abbreviate $\textsc{Pattern}_\Sigma$ as $\textsc{Pattern}$ when Σ is understood.

Remark 4.2. Compared to [Chen and Roşu, 2020a], our syntax in Definition 4.1 adds the μ-binder, which is used to build the least fixpoint pattern $\mu X.\, \varphi$. As we will see later in this paper, it is the key construct that captures initial algebra semantics axiomatically. Our syntax also extends first-order modal μ-calculus [Groote and Mateescu, 1999], which generalizes the classical (propositional) modal μ-calculus with FOL quantification. Since propositional μ-calculus is a fragment of matching logic [Chen and Roşu, 2019a] and FOL quantification is present in the grammar in Eq. (1), first-order modal μ-calculus falls as a methodological fragment of matching logic.

We call $\varphi_1\,\varphi_2$ an *application pattern*, which is left associative. Both $\exists x.\,\varphi$ and $\mu X.\,\varphi$ create the binding of x and, resp., X into φ. The scope of a binder goes as far as possible to the right, so for example, $\exists x.\, y \to x$ should be understood as $\exists x.\,(y \to x)$. We assume the standard notions of *free variables* $FV(\varphi) \subseteq EV \cup SV$, *$\alpha$-equivalence* $\varphi_1 \equiv_\alpha \varphi_2$, and *capture-free substitution* $\varphi[\psi/x]$. We regard α-equivalent patterns to be *syntactically identical*. We define the following notations:

1. $\neg\varphi \equiv \varphi \to \bot$;
2. $\top \equiv \neg\bot$;
3. $\varphi_1 \vee \varphi_2 \equiv \neg\varphi_1 \to \varphi_2$;
4. $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$;
5. $\varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$;
6. $\forall x.\,\varphi \equiv \neg\exists x.\,\neg\varphi$;
7. $\nu X.\,\varphi \equiv \neg\mu X.\,\neg\varphi[\neg X/X]$.

4.2 Matching Logic Semantics

Matching logic patterns are interpreted on an underlying carrier set. A pattern is interpreted as a *set* that includes all the elements that *match* the pattern. Intuitively, the pattern $\bot$ (called bottom) is matched by no elements, while $\top$ (called top) is matched by all elements. Conjunction $\varphi_1 \wedge \varphi_2$ is matched by the elements that match both φ_1 and φ_2, disjunction $\varphi_1 \vee \varphi_2$ by the elements that match φ_1 or φ_2, negation $\neg\varphi$ by the elements that do not match φ, and implication $\varphi_1 \to \varphi_2$ by all elements a such that if a matches φ_1 then a matches φ_2. Element variable x is matched by the element to which x evaluates (see Definition 4.4). Set variable X is matched by the set of elements to which X evaluates; this set can be empty, or total, or any subset of the carrier set. Quantification $\exists x.\,\varphi$ is matched by the elements that match φ for *some* valuation of x; that is, it abstracts away the irrelevant part x from the matched part φ. Least fixpoint $\mu X.\,\varphi$ is matched by the smallest set X of elements that satisfies the equation $X = \varphi$ (this is interesting when X occurs in φ).

Now, we formally define matching logic models and the valuation of patterns.

Definition 4.3. Let Σ be a matching logic signature. A Σ-*model* M, or simply a model, consists of:

1. a nonempty *carrier set*, which we also denote M;
2. an *application interpretation* $\_\bullet\_ \colon M \times M \to \mathcal{P}(M)$, where $\mathcal{P}(M)$ is the powerset; and
3. for every $\sigma \in \Sigma$, a *symbol interpretation* $\sigma_M \subseteq M$ as a subset.

Let us compare matching logic models to FOL models. Both types of models are associated with a nonempty carrier set, but FOL constants are interpreted as elements while matching logic symbols/constants are interpreted as subsets of the carrier set. Similarly, FOL interprets binary operations as functions $M \times M \to M$ that return one element while matching logic interprets the application as a function that returns a set. We use the terminology *functional interpretation* to refer to how FOL interprets functions and terms. Note that the FOL functional interpretation is a special instance of the matching logic *set-theoretic interpretation*: due to the bijection between an element a and the singleton $\{a\}$, any set M is isomorphic to the set of all singletons of M.

Definition 4.4. Let Σ be a matching logic signature and M be a Σ-model. Let $\_\bar{\bullet}\_ \colon \mathcal{P}(M) \times \mathcal{P}(M) \to \mathcal{P}(M)$ be the *pointwise extension* of $\_\bullet\_$, i.e., $A \mathbin{\bar{\bullet}} B = \bigcup_{a\in A, b\in B} a \bullet b$ for $A, B \subseteq M$. A(n) *(M-)valuation* $\rho \colon (EV \cup SV) \to M \cup \mathcal{P}(M)$ is a function that maps element variables to elements and set variables to sets, i.e., $\rho(x) \in M$ for $x \in EV$ and $\rho(X) \subseteq M$ for $X \in SV$. It yields a *pattern valuation*, written $|\_|_{M,\rho} : \textsc{Pattern} \to \mathcal{P}(M)$, which is inductively defined as follows:

1. $|x|_{M,\rho} = \{\rho(x)\}$ for $x \in EV$;
2. $|X|_{M,\rho} = \rho(X)$ for $X \in SV$;
3. $|\sigma|_{M,\rho} = \sigma_M$ for $\sigma \in \Sigma$;
4. $|\varphi_1\, \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \mathbin{\bar{\bullet}} |\varphi_2|_{M,\rho}$;
5. $|\bot|_{M,\rho} = \emptyset$;
6. $|\varphi_1 \to \varphi_2|_{M,\rho} = M \setminus (|\varphi_1|_{M,\rho} \setminus |\varphi_2|_{M,\rho})$;
7. $|\exists x.\, \varphi|_{M,\rho} = \bigcup_{a \in M} |\varphi|_{M,\rho[a/x]}$;
8. $|\mu X.\, \varphi|_{M,\rho} = \bigcap\{A \subseteq M \mid |\varphi|_{M,\rho[A/X]} \subseteq A\}$;

where "$\setminus$" denotes set difference, $\rho[a/x]$ (resp. $\rho[A/X]$) is the updated valuation ρ' such that $\rho'(x) = a$ (resp. $\rho'(X) = A$) and agrees with ρ on all the other (element and set) variables in $EV \cup SV \setminus \{x\}$ (resp. $EV \cup SV \setminus \{X\}$). We abbreviate $|\varphi|_{M,\rho}$ as $|\varphi|_\rho$ when M is understood. When φ is a closed pattern, i.e., $FV(\varphi) = \emptyset$, we abbreviate $|\varphi|_{M,\rho}$ as $|\varphi|_M$ or just $|\varphi|$, because ρ is irrelevant.

Remark 4.5. The above semantic rules are not unexpected. Rules (1) and (2) interpret variables according to ρ. Rules (3) and (4) interpret symbols and application according to M. For rules (5)-(7), if we regard $\emptyset$ as "false" and M as "true", then these rules become the FOL semantic rules of bottom, implication, and $\exists$-quantification, respectively. Rule (8) computes the smallest set A such that $|\varphi|_{M,\rho[A/X]} = A$, by the Knaster-Tarski fixpoint theorem [Tarski, 1955]; see Section 4.2.3.

Proposition 4.6. *The following propositions hold:*

1. $|\neg\varphi|_{M,\rho} = M \setminus |\varphi|_{M,\rho}$*;*
2. $|\varphi_1 \vee \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \cup |\varphi_2|_{M,\rho}$*;*
3. $|\varphi_1 \wedge \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \cap |\varphi_2|_{M,\rho}$*;*
4. $|\top|_{M,\rho} = M$*;*
5. $|\varphi_1 \leftrightarrow \varphi_2|_{M,\rho} = M \setminus (|\varphi_1|_{M,\rho} \triangle |\varphi_2|_{M,\rho})$*;*
6. $|\forall x.\, \varphi|_{M,\rho} = \bigcap_{a \in M} |\varphi|_{\rho[a/x]}$*;*
7. $|\nu X.\, \varphi|_{M,\rho} = \bigcup\{A \subseteq M \mid A \subseteq |\varphi|_{M,\rho[A/X]}\}$*;*

where "$\triangle$" denotes set symmetric difference.

We next discuss three important types of patterns: predicate patterns, functional patterns, and fixpoint patterns. Intuitively, predicate patterns are the counterpart of FOL formulas; they make "statements", so their valuations can only be either true, denoted by total set M, or false, denoted by empty set $\emptyset$. Functional patterns are the matching logic counterpart of FOL terms; they denote elements, so their valuations are always singletons. Fixpoint patterns create induction and recursion.

4.2.1 Predicate Patterns

Unlike FOL, matching logic patterns can be interpreted as any subsets of the carrier set. Following up on Remark 4.5, we identify two special sets, M and $\emptyset$, and use them to represent (logical) true and false, respectively. For a pattern φ, we call it a *predicate (pattern)* if $|\varphi|_{M,\rho} \in \{\emptyset, M\}$ for all M and ρ and say that φ *holds*, if it evaluates to the total carrier set, M. Several important predicates such as equality and membership will be defined as examples in Section 4.3.1.

4.2.2 Functional Patterns

A *functional pattern* φ is one that always evaluates to a singleton set, i.e., for any M and ρ there exists $a \in M$ such that $|\varphi|_{M,\rho} = \{a\}$. Therefore, a functional pattern denotes, or is matched by, exactly one element, so we often blur the semantic distinction between elements and singletons, and regard φ as being the element a. The simplest functional pattern is an element variable x, while more interesting examples can be built from symbols and application. In Section 4.3.1, we will show how to define functional patterns axiomatically. More concrete examples are included in Section 4.3.2 and throughout the paper.

4.2.3 Fixpoint Patterns

Given a model M and a valuation ρ, the fixpoint pattern $\mu X.\, \varphi$ yields a function $\mathcal{F}\colon \mathcal{P}(M) \to \mathcal{P}(M)$, defined by $\mathcal{F}(A) = |\varphi|_{\rho[A/X]}$ for all $A \subseteq M$. By the requirement that φ is positive in X (Definition 4.1), we can prove that $\mathcal{F}$ is a monotone function, whose unique least fixpoint $\mu\mathcal{F} \subseteq M$ is exactly the valuation $|\mu X.\, \varphi|_{M,\rho}$. Therefore, $\mu X.\, \varphi$ is a direct, logical incarnation of the least fixpoints in powersets into matching logic patterns. More interestingly, the semantic rule in Definition 4.4(8) inspires an *induction principle* for reasoning about fixpoint patterns, which we discuss informally here and formalize in Section 4.4 as the proof rule (Knaster-Tarski). By Definition 4.4(8), to show that $|\mu X.\, \varphi|_{M,\rho} \subseteq A$ for a set A, one only needs to show that $|\varphi|_{M,\rho[A/X]} \subseteq A$; or written syntactically, to prove that $(\mu X.\, \varphi) \to \psi_A$ for a pattern/property ψ_A, one only needs to prove that $\varphi[\psi_A/X] \to \psi_A$. This general form of inductive reasoning is included in Section 4.4.

4.3 Matching Logic Specifications

Definition 4.7. Let Σ be a matching logic signature and M be a Σ-model. For a Σ-pattern φ, we say that M *validates* φ or φ *holds* in M, written $M \vDash \varphi$, if $|\varphi|_{M,\rho} = M$ for all valuations ρ. Let Γ be a set of Σ-patterns called *axioms*. We say that M *validates* Γ or M is a Γ*-model*, if $M \vDash \varphi$ for all $\varphi \in \Gamma$. For a Σ-pattern ψ, if $M \vDash \psi$ for all models $M \vDash \Gamma$, then we write $\Gamma \vDash \psi$. A *matching logic specification* consists of a matching logic signature Σ and an axiom set Γ of Σ-patterns.

Below, we give two examples of matching logic specifications. The first example in Section 4.3.1 shows how to axiomatically capture the *true equality* in any model M, by which we mean the identity relation over M, and not any equivalence or congruence relation. Recall that FOL can *not* capture the true equality but only congruence relations, which is why FOL has been extended to FOL with equality that adds explicit syntactic and semantic components to support true equality; see, e.g. [Hamilton, 1978, Definition 5.7]. In contrast, we will see that true equality can be axiomatically defined using one symbol and one axiom in matching logic, without the need to extend the logic. Besides, the true equality is used to define other important mathematical instruments such as membership and set inclusion, all of which are examples of predicate patterns (Section 4.2.1).

The second example in Section 4.3.2 defines the set of natural numbers as the smallest set built from zero and the successor function. This example is interesting and important for several reasons. It shows how to axiomatically define functions and functional patterns (Section 4.2.2). It shows how to axiomatically define inductive structures, such as natural numbers, using fixpoint patterns (Section 4.2.3). It also forms the prelude specification given in Section 5, which will be imported and used in the specifications of initial algebra semantics in Section 7.

4.3.1 The First Example: Capturing Equality

Here we define the specification that captures equality. Specifically, we define a predicate pattern $\varphi = \varphi'$ that holds iff φ equals to φ' in all models. To do that, we introduce an important mathematical instrument called *definedness*. A definedness pattern $\lceil \varphi \rceil$ is a predicate pattern that holds iff φ is *defined*, that is, it is matched by at least one element.

Definition 4.8. We define $\mathsf{DEFINEDNESS}$ in Specification. 1.

For clarity, we propose and use a semi-formal syntax for writing specification definitions. A specification, like DEFINEDNESS, is enclosed in the keywords **spec** and **endspec** and has a name DEFINEDNESS, all in capital letters. Within it are several declaration keywords: Symbols declares a list of matching logic symbols; Notations defines a list of notations, where "≡" is read as "is sugar for"; Axioms defines a list of named axioms. There is a fourth declaration keyword called Imports that is used to import existing specifications. We will see one example in Specification 2 shortly.

```
spec DEFINEDNESS
  Symbols: $def$
  Notations:
    $\lceil \varphi \rceil \equiv def\ \varphi$
  Axioms:
    (Definedness)  $\forall x.\ \lceil x \rceil$
endspec
```

Specification 1: DEFINEDNESS

```
spec DEFINEDNESS$^+$
  Imports: DEFINEDNESS
  Notations:
    $\lfloor \varphi \rfloor \equiv \neg\lceil \neg\varphi \rceil$                      // totality
    $\varphi_1 = \varphi_2 \equiv \lfloor \varphi_1 \leftrightarrow \varphi_2 \rfloor$   // (true) equality
    $\varphi_1 \subseteq \varphi_2 \equiv \lfloor \varphi_1 \to \varphi_2 \rfloor$   // set inclusion
    $x \in \varphi \equiv x \subseteq \varphi$                 // membership
endspec
```

Specification 2: DEFINEDNESS$^+$

DEFINEDNESS defines one symbol def, one notation $\lceil \varphi \rceil$, and one axiom $\forall x.\ \lceil x \rceil$ named (Definedness). (Definedness) states that every element x is defined. This is in tune with our intuition about definedness—matching at least one element—because, indeed, x is matched by exactly one element, which is the one to which it evaluates. Now, consider any model $M \vDash$ DEFINEDNESS and any pattern φ. We have two cases: (1) φ is defined in M, or (2) φ is undefined. If it is defined, then there exists at least an element x included in φ. By pointwise extension (Definition 4.7), $\lceil x \rceil$ is included in $\lceil \varphi \rceil$, and since $\lceil x \rceil$ holds by (Definedness), $\lceil \varphi \rceil$ must also hold. If φ is undefined, then φ is the empty set. Also by pointwise extension, $\lceil \varphi \rceil$ must also be empty. Thus, $\lceil \varphi \rceil$ is a predicate that holds iff φ is defined. Formally,

Proposition 4.9. *For any $M \vDash$ DEFINEDNESS, pattern φ, and valuation ρ:*

1. *$\lceil a \rceil_M = M$ for every $a \in M$, where $\lceil a \rceil_M = def_M \mathbin{\bar\cdot} \{a\}$ and def_M is the interpretation of def;*
2. *$|\lceil \varphi \rceil|_{M,\rho} = M$ if $|\varphi|_{M,\rho} \neq \emptyset$; otherwise, $|\lceil \varphi \rceil|_{M,\rho} = \emptyset$.*

We use definedness to define equality and also totality, membership, and inclusion in DEFINEDNESS$^+$, Specification 2. Their semantic meaning is formalized by the following proposition. Note that semantically, $x \in \varphi$ is the same as $x \subseteq \varphi$. We still define $x \in \varphi$ because it fits well the intuition that x is an element in φ.

Proposition 4.10. *For $M \vDash$ DEFINEDNESS, patterns φ, φ', element variable x, and valuation ρ:*

1. *$|\lfloor \varphi \rfloor|_{M,\rho} = M$ if $|\varphi|_{M,\rho} = M$; otherwise, $|\lfloor \varphi \rfloor|_{M,\rho} = \emptyset$;*
2. *$|\varphi = \varphi'|_{M,\rho} = M$ if $|\varphi|_{M,\rho} = |\varphi'|_{M,\rho}$; otherwise, $|\varphi = \varphi'|_{M,\rho} = \emptyset$;*
3. *$|\varphi \subseteq \varphi'|_{M,\rho} = M$ if $|\varphi|_{M,\rho} \subseteq |\varphi'|_{M,\rho}$; otherwise, $|\varphi \subseteq \varphi'|_{M,\rho} = \emptyset$;*
4. *$|x \in \varphi|_{M,\rho} = M$ if $\rho(x) \in |\varphi|_{M,\rho}$; otherwise, $|x \in \varphi|_{M,\rho} = \emptyset$.*

4.3.2 The Second Example: Capturing Natural Numbers

Here we define NN in Specification 3 that captures the set $\mathbb{N}$ of natural numbers as the smallest set built from zero and the successor function.

NN imports DEFINEDNESS$^+$ and defines three symbols: $\mathbb{N}$ denotes the set of natural numbers; *zero* denotes the number zero; and *succ* denotes the successor function. Intuitively, axiom (Nat Zero) is a typical axiom, which states that *zero* is a functional pattern (see Section 4.2.2), and the element that *zero* denotes is in $\mathbb{N}$ (recall that x is matched by exactly one element). Axiom (Nat Succ) states that for any x in $\mathbb{N}$, ($succ\ x$) is an element in $\mathbb{N}$. Axioms (Nat Succ.1) and (Nat Succ.2) state that *zero*, ($succ\ zero$), ($succ\ succ\ zero$), . . . are all distinct. Finally, (Nat Domain) defines $\mathbb{N}$ as the smallest set D that includes *zero* and is closed under *succ*.

The above intuition about defining functional patterns and functions in matching logic is of vital importance to understand the specifications of initial algebra semantics in later sections. Therefore, we give the formal details below. Let us consider a model $M \vDash \mathsf{NN}$, where $\_\bullet\_ \colon M \times M \to \mathcal{P}(M)$ is the application interpretation, $\_\bar{\bullet}\_$ is its pointwise extension, and $\mathbb{N}_M, zero_M, succ_M \subseteq M$ are the corresponding symbol interpretations. Then, M derives the standard model of natural numbers as follows, where for clarity we explicitly specify the axioms that make the claims true:

```
spec NN
  Imports: DEFINEDNESS+
  Symbols: ℕ, zero, succ
  Axioms:
    (Nat Zero)     ∃x. x ∈ ℕ ∧ zero = x
    (Nat Succ)     ∀x. x ∈ ℕ → ∃y. y ∈ ℕ ∧ succ x = y
    (Nat Succ.1)   succ zero ≠ zero
    (Nat Succ.2)   ∀x. ∀y. x ∈ ℕ ∧ y ∈ ℕ
                      → succ x = succ y → x = y
    (Nat Domain)   ℕ = μD. zero ∨ succ D
endspec
```

Specification 3: NN

Proposition 4.11. *For any $M \vDash \mathsf{NN}$, the following properties hold:*

1. *By axiom* (Nat Zero)*: $\mathbb{N}_M \neq \emptyset$; $zero_M$ is a singleton; and $zero_M \subseteq \mathbb{N}_M$; thus, we can define $M_{zero} \in \mathbb{N}_M$ to be the unique element that is in $zero_M$;*

2. *By axiom* (Nat Succ)*: for any $m \in \mathbb{N}_M$ there exists $next_m \in \mathbb{N}_M$ such that $succ_M \bar{\bullet} \{m\} = \{next_m\}$; thus, we can define $M_{succ} \colon \mathbb{N}_M \to \mathbb{N}_M$ to be the unique function such that $M_{succ}(m) = next_m$;*

3. *By axioms* (Nat Succ.1) *and* (Nat Succ.2)*: $M_{succ}(M_{zero}) \neq M_{zero}$, and for any $m, n \in \mathbb{N}_M$, $M_{succ}(m) = M_{succ}(n)$ implies $m = n$, that is, M_{succ} is injective;*

4. *By axiom* (Nat Domain)*: $\mathbb{N}_M$ is the set $\{M_{zero}, M_{succ}(M_{zero}), M_{succ}(M_{succ}(M_{zero})), \dots\}$;*

5. *Thus, $(\mathbb{N}_M, M_{zero}, M_{succ})$ is the standard model of natural numbers.*

As a notational convention, we write $zero_M$ and $succ_M$ for the symbol interpretations given directly by the matching logic model, and M_{zero} and M_{succ} for the elements and/or functions that are derived from M as above. Similarly, in Section 5 we will define sorts in matching logic, and we will write s_M for the interpretation of the sort name s given directly by the matching logic model, and M_s for the actual inhabitant set of s derived from M. This way, our matching logic notations are the same as the notations for carrier sets and operation interpretations of algebras (Definition 3.3).

4.4 Matching Logic Proof System

Matching logic has a *fixed* Hilbert-style proof system that supports formal reasoning for all specifications Γ. We write $\Gamma \vdash \varphi$ to mean that φ is a provable pattern from the axioms in Γ. In this paper, the actual proof system is not necessary for the technical results, so we exile it to Appendix B. In the following, we review some important meta-theorems that can be proved about the proof system so as to give intuition about the types of formal reasoning that are supported in matching logic.

Proposition 4.12. *Let Γ be any specification. Then, the following propositions hold:*

1. *$\Gamma \vdash \varphi$, if φ is a tautology over patterns;*

2. *$\Gamma \vdash \varphi_1$ and $\Gamma \vdash \varphi_1 \to \varphi_2$ imply $\Gamma \vdash \varphi_2$;*

3. *$\Gamma \vdash \varphi[y/x] \to \exists x. \varphi$;*

4. *$\Gamma \vdash \varphi_1 \to \varphi_2$ and $y \notin FV(\varphi_2)$ imply $\Gamma \vdash (\exists y. \varphi_1) \to \varphi_2$;*

5. *$\Gamma \vdash \varphi = \varphi$;*

6. *$\Gamma \vdash \varphi_1 = \varphi_2$ and $\Gamma \vdash \varphi_2 = \varphi_3$ imply $\Gamma \vdash \varphi_1 = \varphi_3$;*

7. $\Gamma \vdash \varphi_1 = \varphi_2$ *implies* $\Gamma \vdash \varphi_2 = \varphi_1$*;*

8. $\Gamma \vdash \varphi_1 = \varphi_2$ *implies* $\Gamma \vdash \psi[\varphi_1/x] = \psi[\varphi_2/x]$*, known as the Leibniz's law of equality.*

9. $\Gamma \vdash (\mu X.\, \varphi) = \varphi[\mu X.\, \varphi/X]$*;*

10. $\Gamma \vdash \varphi[\psi/X] \to \psi$ *implies* $\Gamma \vdash (\mu X.\, \varphi) \to \psi$*; this proof rule is denoted* (Knaster-Tarski)*;*

where for (5)-(9) we naturally require that Γ *defines equality (Section 4.3.1).*

Properties (1)-(4) capture standard FOL reasoning, (5)-(8) capture standard equational reasoning, and (9)-(10) provide standard fixpoint reasoning (cf. [Kozen, 1983, Section 5]). Particularly, rule (9) states that $\mu X.\, \varphi$ is a fixpoint, and thus it is the same after unfolding. Rule (10) characterizes an induction principle about $\mu X.\, \varphi$, following the discussion in Section 4.2.3. As expected, $\vdash$ is sound:

Theorem 4.13. *For any* Γ *and* φ*, we have* $\Gamma \vdash \varphi$ *implies* $\Gamma \vDash \varphi$*.*

5 Capturing Sorts: Pairs, Tuples, and Functions

Matching logic is unsorted. In this section, we propose a systematic and extensible way to defining arbitrary sorting structures axiomatically in matching logic and show in detail how to define pair sorts, tuple sorts, and function sorts as examples. We only discuss these very basic sort constructs here, since they are all we need to capture many-sorted initial algebra. More complex sort constructs, such as subsorts, parametric sorts, and recursively-constrained sorts are discussed in Section 10.

5.1 Sorts in Matching Logic

By Definition 4.3, a matching logic model M has only one universal carrier set. As seen from specification NN of natural numbers (Section 4.3.2), the carrier set includes not only the intended elements but also elements corresponding to symbols meant to be functions, predicates, etc. From that perspective, in matching logic a sort s is regarded as a means to refer to a *subset* of the total universe M including only the elements "having sort s". Specifically, we can define a sort s as a symbol that represents the *name* of the sort, which we use to generically refer to all the elements that inhabit it. To obtain the actual subset of elements associated with s, called the *inhabitant set of s*, we define a symbol *inh*, called the *inhabitant symbol*, and use the application pattern (*inh* s) to represent the inhabitant set of s.

```
  spec SORT
  Imports: DEFINEDNESS⁺
  Symbols: inh, Sort
  Notations:
    $[\![s]\!] \equiv inh\ s$
    $\neg_s\varphi \equiv (\neg\varphi) \wedge [\![s]\!]$
    $\forall x{:}s.\, \varphi \equiv \forall x.\, x \in [\![s]\!] \to \varphi$
    $\exists x{:}s.\, \varphi \equiv \exists x.\, x \in [\![s]\!] \wedge \varphi$
    $\varphi{:}s \equiv \exists z{:}s.\, \varphi = z$
    $\forall x_1, \ldots, x_n{:}s.\, \varphi \equiv \forall x_1{:}s.\, \ldots \forall x_n{:}s.\, \varphi$
    $\exists x_1, \ldots, x_n{:}s.\, \varphi \equiv \exists x_1{:}s.\, \ldots \exists x_n{:}s.\, \varphi$
endspec
```

Specification 4: SORT

Formally, we define SORT in Specification 4, which provides no particular sorts are defined but only the generic sorting infrastructure, which supports the subsequent sort structures such as pair sorts, function sorts, etc. Specifically, SORT defines two symbols *inh* and *Sort*, and some standard notations about sorting. As mentioned, *inh* is the inhabitant symbol, and pattern (*inh* s), also written $[\![s]\!]$ according to the notation, is matched by all elements that have sort s. *Sorted negation* $\neg_s\varphi$ is matched by the elements of sort s that do not match φ. *Sorted quantification* $\forall x{:}s.\, \varphi$ and $\exists x{:}s.\, \varphi$ restrict the range of x to the inhabitants of s. *Sorted membership* $\varphi{:}s$ specifies that φ is a functional pattern (Section 4.2.2) and that φ is an inhabitant of s. Symbol *Sort* is the sort of all sort names. Later in this section, we will define pair sorts, tuple sorts, and function sorts for all sorts in *Sort*, which thus act as *constructors* for *Sort*.

Remark 5.1. SORT is a generic specification to be imported when sorts are desired. The actual semantic meaning of an element x having a sort s is arbitrary and open for interpretation, and is completely decided by (user-defined) axioms. For example, if *Nat* is the sort of natural numbers (see Section 5.1.1), then x having sort *Nat* means that x is a natural number. In general, a sort s can mean that a certain property holds, and then the sorted membership $x{:}s$ holds iff x has the property associated with s, which is reminiscent of the typing relation $t{:}\tau$ in type systems.

5.1.1 An Example: The Sort of Natural Numbers

We defined a specification NN of natural numbers (Specification 3). Based on that, we define the sort *Nat* of natural numbers in NAT as Specification 5.

Intuitively, (Nat Sort) states that *Nat* is an inhabitant of *Sort*. (Nat) states that the inhabitant set of *Nat* equals $\mathbb{N}$, which is the set of natural numbers defined in Section 4.3.2. The other four axioms define the Peano addition and multiplication of natural numbers, where we use the sorted quantification (Specification 4).

NAT is an important example that shows how to use the generic sorting infrastructure in SORT to axiomatically define sorts and operations. The sort *Nat* of natural numbers will also be imported and used by the specification of tuples (Section 5.3) to define tuple projection.

spec NAT
  Imports: NN, SORT
  Symbols: $Nat, plus, mult$
  Axioms:

(Nat Sort)	$Nat{:}Sort$
(Nat)	$[\![Nat]\!] = \mathbb{N}$
(Nat Plus.1)	$\forall x{:}Nat.\, plus\ x\ zero = x$
(Nat Plus.2)	$\forall x, y{:}Nat.\, plus\ x\ (succ\ y) = succ\ (plus\ x\ y)$
(Nat Mult.1)	$\forall x{:}Nat.\, mult\ x\ zero = zero$
(Nat Mult.2)	$\forall x, y{:}Nat.\, mult\ x\ (succ\ y) = plus\ x\ (mult\ x\ y)$

endspec

Specification 5: NAT

Like in Section 4.3.2, we can consider a model $M \vDash$ NAT and show how symbol interpretations $plus_M, mult_M \subseteq M$ derive the standard addition and multiplication M_{plus} and M_{mult} over naturals, same way the successor function M_{succ} was derived. Specifically, if $M_{Sort} = |[\![Sort]\!]|_M = inh_M \,\bar{\bullet}\, Sort_M$ as is the inhabitant set of *Sort* and $M_{Nat} = |[\![Nat]\!]|_M = inh_M \,\bar{\bullet}\, Nat_M$ is the inhabitant set of *Nat*, then,

Proposition 5.2. *The following propositions hold:*

1. *By* (Nat Sort)*: Nat_M is a singleton, whose unique element we (ambiguously) denote also as Nat_M; then, $Nat_M \in M_{Sort}$; intuitively, Nat_M is the interpretation of the sort name Nat in M;*
2. *By* (Nat)*: the set M_{Nat} equals $\mathbb{N}_M = \{M_{zero}, M_{succ}(M_{zero}), M_{succ}(M_{succ}(M_{zero})), \dots\}$, which was defined in Proposition 4.11; intuitively, M_{Nat} is the inhabitant set of Nat in M;*
3. *Following the same reasoning in Proposition 4.11, we can define functions $M_{plus}, M_{mult} \colon M_{Nat} \times M_{Nat} \to M_{Nat}$, such that $plus_M \,\bar{\bullet}\, \{m\} \,\bar{\bullet}\, \{n\} = \{M_{plus}(m, n)\}$ and $mult_M \,\bar{\bullet}\, \{m\} \,\bar{\bullet}\, \{n\} = \{M_{mult}(m, n)\}$, for all $m, n \in M_{Nat}$; that is, they capture the addition and multiplication functions.*

In other words, M_{Sort} is the inhabitant set of *Sort* and $Nat_M \in M_{Sort}$ denotes the sort name *Nat*. M_{Nat} is the inhabitant set of *Nat*, and $M_{zero}, M_{succ}, M_{plus}, M_{mult}$ are element and/or functions over M_{Nat}.

For notational simplicity, we want to use natural numbers $0, 1, 2, \dots$ *as is* in matching logic patterns and specifications. Thus, we define NAT$^+$ in Specification 6 that defines all natural numbers $0, 1, 2, \dots$ as notations that are desugared into the corresponding patterns *zero*, (*succ zero*), (*succ* (*succ zero*)), ... We also define a new sort *NzNat* for positive (nonzero) natural numbers as a *subsort* of *Nat*; Section 10 shows how subsorts can be rigorously handled in matching logic.

spec NAT$^+$
  Imports: NAT
  Symbols: $NzNat$
  Notations:
    $0 \equiv zero$
    $1 \equiv succ\ 0$
    $2 \equiv succ\ 1$
    $\cdots$
  Axioms:

(NzNat Sort)	$NzNat{:}Sort$
(NzNat)	$[\![NzNat]\!] = succ\ [\![Nat]\!]$

endspec

Specification 6: NAT$^+$

5.2 Pair Sorts

For any two sorts s_1 and s_2, we define a new sort *Pair* s_1 s_2, called the *pair sort of* s_1 *and* s_2. Here, *Pair* is a symbol that serves as a *sort constructor*. For the new pair sort, we also define a symbol *pair* as the constructor of the pairs and *fst*, *snd* destructors. This is formalized by PAIR in Specification 7.

spec PAIR
Imports: SORT
Symbols: $\mathit{Pair}, \mathit{pair}, \mathit{fst}, \mathit{snd}$
Notations:
$s_1 \otimes s_2 \equiv \mathit{Pair}\ s_1\ s_2$
$\langle \varphi_1, \varphi_2 \rangle \equiv \mathit{pair}\ \varphi_1\ \varphi_2$
Axioms: // all axioms are quantified by "$\forall s_1, s_2 : \mathit{Sort}$"

(Pair Sort)	$(s_1 \otimes s_2) : \mathit{Sort}$
(Pair)	$\forall x_1 : s_1 . \forall x_2 : s_2 . \ \langle x_1, x_2 \rangle : (s_1 \otimes s_2)$
(Pair Fst)	$\forall x_1 : s_1 . \forall x_2 : s_2 . \mathit{fst}\ \langle x_1, x_2 \rangle = x_1$
(Pair Snd)	$\forall x_1 : s_1 . \forall x_2 : s_2 . \mathit{snd}\ \langle x_1, x_2 \rangle = x_2$
(Pair Inj)	$\forall x_1, y_1 : s_1 . \forall x_2, y_2 : s_2 .$ $\langle x_1, x_2 \rangle = \langle y_1, y_2 \rangle \to x_1 = x_2 \wedge y_1 = y_2$
(Pair Domain)	$[\![s_1 \otimes s_2]\!] = \langle [\![s_1]\!], [\![s_2]\!] \rangle$

endspec

Specification 7: PAIR

Intuitively, $\langle x_1, x_2 \rangle$ is the pair of x_1 and x_2. If x_1 has sort s_1 and x_2 has sort s_2, then by (Pair), $\langle x_1, x_2 \rangle$ is an element of sort $s_1 \otimes s_2$. (Pair Sort) states that $s_1 \otimes s_2$ is a sort when s_1 and s_2 are sorts in *Sort*. Hence, we can have nested product sorts such as $s_1 \otimes (s_2 \otimes s_3)$ where s_1, s_2, s_3 are sorts in *Sort*. This allows us to define (finite) *tuple sorts* (Section 5.3). (Pair Fst) and (Pair Snd) define the destructors *fst* and *snd*. (Pair Inj) states that two pairs are equal only if their corresponding components are equal. Finally, (Pair Domain) states that the inhabitant set of $s_1 \otimes s_2$ is the set of all pairs $\langle x_1, x_2 \rangle$, with x_1 of sort s_1 and x_2 of sort s_2.

Like in Propositions 4.11 and 5.2, we formalize the above intuition by considering a model $M \vDash$ PAIR, where $\mathit{pair}_M, \mathit{fst}_M, \mathit{snd}_M, \mathit{Pair}_M \subseteq M$ are the corresponding symbol interpretations. Recall that we let $M_{\mathit{Sort}} = M_{\mathit{inh}} \mathbin{\bar{\bullet}} \mathit{Sort}_M$ to be the inhabitant set of *Sort* that includes all sort names in M. Then for each $s \in M_{\mathit{Sort}}$, let the set $M_s = M_{\mathit{inh}} \mathbin{\bar{\bullet}} \{s\}$ be the inhabitant set of s in M.

Proposition 5.3. *Under the above conditions and notations, the following properties hold:*

1. *By* (Pair Sort)*: for $s_1, s_2 \in M_{\mathit{Sort}}$, $\mathit{Pair}_M \mathbin{\bar{\bullet}} \{s_1\} \mathbin{\bar{\bullet}} \{s_2\}$ is a singleton, whose (unique) element we denote $s_1 \otimes_M s_2$; then we have $s_1 \otimes_M s_2 \in M_{\mathit{Sort}}$; intuitively, $s_1 \otimes_M s_2$ is the pair sort of s_1 and s_2;*
2. *By* (Pair)*: for $s_1, s_2 \in M_{\mathit{Sort}}$, $x_1 \in M_{s_1}$, and $x_2 \in M_{s_2}$, $\mathit{pair}_M \mathbin{\bar{\bullet}} \{x_1\} \mathbin{\bar{\bullet}} \{x_2\}$ is a singleton, whose (unique) element we denote $\langle x_1, x_2 \rangle_M$; then we have $\langle x_1, x_2 \rangle_M \in M_{s_1 \otimes_M s_2}$;*
3. *Like in Propositions 4.3.2 and 5.1.1, let $M_{\mathit{fst}} \colon M_{s_1 \otimes_M s_2} \to M_{s_1}$ and $M_{\mathit{snd}} \colon M_{s_1 \otimes_M s_2} \to M_{s_2}$ be such that $\mathit{fst}_M \mathbin{\bar{\bullet}} \{y\} = \{M_{\mathit{fst}}(y)\}$ and $\mathit{snd}_M \mathbin{\bar{\bullet}} \{y\} = \{M_{\mathit{snd}}(y)\}$, for any $s_1, s_2 \in M_{\mathit{Sort}}$ and $y \in M_{s_1 \otimes_M s_2}$;*
4. *By* (Pair Fst/Snd/Inj)*: for $s_1, s_2 \in M_{\mathit{Sort}}$, $x_1, y_1 \in M_{s_1}$, and $x_2, y_2 \in M_{s_2}$, we have $M_{\mathit{fst}}(\langle x_1, x_2 \rangle_M) = x_1$, $M_{\mathit{snd}}(\langle x_1, x_2 \rangle_M) = x_2$, and $\langle x_1, x_2 \rangle_M = \langle y_1, y_2 \rangle_M$ implies $x_i = y_i$, $(i = 1, 2)$;*
5. *$M_{s_1 \otimes_M s_2} = \{ \langle x_1, x_2 \rangle_M \mid x_1 \in M_{s_1}, x_2 \in M_{s_2} \}$;*
6. *Then, $M_{s_1 \otimes_M s_2}$ is exactly the Cartesian product $M_{s_1} \times M_{s_2}$ of M_{s_1} and M_{s_2}, for $s_1, s_2 \in M_{\mathit{Sort}}$.*

5.3 Tuple Sorts

Tuples are nested pairs, so we import PAIR and define notations for tuples. We also define a new symbol *proj*, which takes a positive number i and a tuple, and returns its ith component.

spec TUPLE Imports: PAIR, NAT
Symbols: *proj*
Notations:
$\langle \varphi_1, \varphi_2, \dots, \varphi_n \rangle \equiv \langle \varphi_1, \langle \varphi_2, \dots, \varphi_n \rangle \rangle$
$s_1 \otimes s_2 \otimes \cdots \otimes s_n \equiv s_1 \otimes (s_2 \otimes \cdots \otimes s_n)$
Axioms: // all axioms are quantified by "$\forall s_1, s_2 : \mathit{Sort}$"

(Proj First)	$\forall x_1 : s_1 . \forall x_2 : s_2 . (\mathit{proj}\ 1\ \langle x_1, x_2 \rangle) = x_1$
(Proj Rest)	$\forall x_1 : s_1 . \forall x_2 : s_2 . \forall n : \mathit{NzNat} .$ $(\mathit{proj}\ (\mathit{succ}\ n)\ \langle x_1, x_2 \rangle) = \mathit{proj}\ n\ x_2$

endspec

Specification 8: TUPLE

Remark 5.4. Following the notations in Proposition 5.3, for a model $M \vDash$ TUPLE, we write $\langle x_1, \dots, x_n \rangle_M \in M_{s_1 \otimes_M \cdots \otimes_M s_n}$ for the tuple of $x_1 \in M_{s_1}$, ..., $x_n \in M_{s_n}$. Then, $M_{s_1 \otimes_M \cdots \otimes_M s_n}$ is exactly the Cartesian product $M_{s_1} \times \cdots \times M_{s_n}$ of sets $M_{s_1}, \dots, M_{s_n}$, for $s_1, \dots, s_n \in M_{\mathit{Sort}}$.

arity of f	**function sort**	**axiomatizing that f is a function**	**applying f on argument(s)**
nullary	$Unit \ominus s$	$f : Unit \ominus s$	$f()$, which equals f
unary	$s_1 \ominus s$	$f : s_1 \ominus s$	$f(x_1)$
multary	$s_1 \otimes \cdots \otimes s_n \ominus s$	$f : s_1 \otimes \cdots \otimes s_n \ominus s$	$f(x_1, \ldots, x_n)$

Table 1: Handling functions in matching logic.

5.4 Function Sorts

For any two sorts s_1 and s_2, we define a new sort *Function* s_1 s_2 called the *function sort from* s_1 *to* s_2, where *Function* is a symbol that serves as a sort constructor. This is formalized by FUNCTION in Specification 9.

Intuitively, (Func Sort) states that $s_1 \ominus s_2$ is a sort in *Sort* whenever s_1 and s_2 are sorts in *Sort*. (Func Domain) states that $s_1 \ominus s_2$ is the sort of all "elements" f that behave as a function from s_1 to s_2, i.e., for any x of sort s_1, the application $(f\ x)$ returns a value of sort s_2. (Func Ext) states that two functions f and g of sort $s_1 \ominus s_2$ are equal iff they return the same value on all arguments of sort s_1.

Although FUNCTION only defines the sorts of unary functions, the same methodology can be applied to dealing with multary functions and nullary functions (i.e., constants) in a uniform way. Specifically, a multary function f from sorts $s_1, \ldots, s_n$ to s for $n \geq 2$, can be defined as a unary function from the pair/tuple sort $s_1 \otimes \cdots \otimes s_n$ to s. A nullary function c of sort s can be defined as a unary function from the unit sort *Unit* to s, where *Unit* is a special sort with exactly one element *unit* that is the right identity of application, i.e., $(c\ unit)$ always equals c. Then, the nullary function c can be represented as a function from *Unit* to s. This leads us to PRELUDE in Specification 10.

spec FUNCTION
Imports: SORT
Symbols: *Function*
Notations: $s_1 \ominus s_2 \equiv Function\ s_1\ s_2$
Axioms: // all axioms are quantified by "$\forall s_1, s_2 : Sort$"

(Func Sort)	$(s_1 \ominus s_2) : Sort$
(Func Domain)	$[\![s_1 \ominus s_2]\!] = \exists f.\, f \wedge (\forall x : s_1.\, (f\ x) : s_2)$
(Func Ext)	$\forall f, g : s_1 \ominus s_2.\, (\forall x : s_1.\, f\ x = g\ x) \to f = g$

endspec

Specification 9: FUNCTION

spec PRELUDE
Imports: FUNCTION, TUPLE
Symbols: *Unit*, *unit*
Notations:
$() \equiv unit$
$f(\varphi) \equiv f\ \varphi$
$f(\varphi_1, \ldots, \varphi_n) \equiv f\ \langle \varphi_1, \ldots, \varphi_n \rangle$ // for $n \geq 2$
Axioms:

(Unit Sort)	$Unit : Sort$
(Unit)	$unit : Unit$
(Unit Domain)	$[\![Unit]\!] = unit$
(Unit Identity)	$\forall s : Sort.\, \forall x : s.\, (x\ unit) = x$

endspec

Specification 10: PRELUDE

PRELUDE shows a systematic way to represent functions of any arity, including nullary, unary, and multary functions, using a uniform and familiar notation as summarized in Table 1.

In the rest of the paper, when we discuss functions, we feel free to mention only the multary cases, where the nullary and unary cases are implicitly covered in the sense described above. For example, when we say that $f : s_1 \otimes \cdots \otimes s_n \ominus s$ is a function, it should be understood that f can be a nullary function (when $n = 0$), or a unary function (when $n = 1$), or a multary function (when $n \geq 2$).

The following proposition characterizes the behavior of a function.

Proposition 5.5. *The following holds for any* $n \geq 0$*:*

$$\mathsf{PRELUDE} \vdash \forall s_1, \ldots, s_n : Sort.\ (f : s_1 \otimes \cdots \otimes s_n \ominus s) \to (\forall x_1 : s_1. \ldots \forall x_n : s_n.\, f(x_1, \ldots, x_n) : s)$$

Remark 5.6. A common design pattern that will occur in the rest of the paper when we define the specifications of initial algebra semantics, is that we declare a symbol f and the following axiom:

$$\text{(Function)} \qquad f : s_1 \otimes \cdots \otimes s_n \ominus s$$

for sorts $s_1, \ldots, s_n, s$ in *Sort*. Following the same idea as in Propositions 4.11 and 5.2, we can show that given a model M, the symbol f indeed derives a function from the inhabitant sets of $s_1, \ldots, s_n$ to the inhabitant set of s in M. Due to its importance, we formalize the intuition in detail below:

Proposition 5.7. *Let* SPEC *be a specification that imports* PRELUDE *and includes the above axiom* (Function) *for a symbol f. Then f yields a function $M_f \colon M_{s_1} \times \cdots \times M_{s_n} \to M_s$ in any $M \vDash$* SPEC.

Note that the proposition holds for all $n \geq 0$ and f can be a nullary, unary, or multary function.

6 Capturing Algebras and Equational Specifications

Here, we define the matching logic specifications that capture (many-sorted) algebras (Definition 3.3) and equational specifications (Definition 3.7). For each signature (S, F), we define a corresponding matching logic specification $\mathsf{ALGEBRA}(S, F)$, where the sorts $s_1, s_2, \dots \in S$ are captured by matching logic symbols of sort *Sort* (Specification 4), and each operation $f \in F_{s_1 \ldots s_n, s}$ is captured by a matching logic symbol f with the (Function) axiom $f : s_1 \otimes \cdots \otimes s_n \ominus s$ (Remark 5.6). To formally show that $\mathsf{ALGEBRA}(S, F)$ indeed captures the (S, F)-algebras, we use the categorical notion of *institution (co)morphisms* [Goguen and Rosu, 2002] to show that there exists a *simple theoroidal comorphism* from the category of (S, F)-algebras to the category of matching logic $\mathsf{ALGEBRA}(S, F)$-models.

For presentational purpose, we will not use much category theory in the main text. Instead, we show the equivalence result using a *model transformation*. Specifically, we define a transformation α that sends a model $M \vDash \mathsf{ALGEBRA}(S, F)$ to an (S, F)-algebra $\alpha(M)$ and prove that M and $\alpha(M)$ validate the same set of F-equations (via syntactic sugar where F-equations are patterns):

$$M \vDash e \quad \text{if and only if} \quad \alpha(M) \vDash_{\mathsf{Alg}} e \tag{2}$$

We also show that α is *essentially surjective*, that is, for any (S, F)-algebra A there exists a model M such that $\alpha(M)$ is exactly A. The reader familiar with the theory of institutions will notice that the above model translation is actually what establishes the above-mentioned simple theoroidal comorphism; for details we refer to Appendix D.3.

6.1 Capturing Signatures

Definition 6.1. For a signature (S, F) like in Definition 3.1, we define $\mathsf{ALGEBRA}(S, F)$ in Specification 11, often abbreviated $\mathsf{ALGEBRA}(F)$.

For technical convenience, we define the auxiliary sorts *SigOps* and *SigArgs*, where *SigOps* is the sort for all operations in F as given by the signature (S, F), and *SigArgs* is the sort of all argument tuples of the operations in F. With these sorts, we can quantify over operations and their arguments using the sorted quantification such as $\forall f : \mathit{SigOps}$ and $\forall \mathit{arg} : \mathit{SigArgs}$.

spec $\mathsf{ALGEBRA}(S, F)$ Imports: PRELUDE
Symbols: $s \in S, f \in F, \mathit{SigOps}, \mathit{SigArgs}$
Axioms:
(Sort) $s : \mathit{Sort}$ for $s \in S$
(Function) $f : s_1 \otimes \cdots \otimes s_n \ominus s$ for $f \in F_{s_1 \ldots s_n, s}$
(Signature Ops) $[\![\mathit{SigOps}]\!] = \bigvee_{f \in F} f$
(Signature Args) $[\![\mathit{SigArgs}]\!] = \bigvee_{f \in F_{s_1 \ldots s_n, s}} [\![s_1 \otimes \cdots \otimes s_n]\!]$
endspec

Specification 11: $\mathsf{ALGEBRA}(S, F)$

6.2 Capturing Algebras

Here we show that specification $\mathsf{ALGEBRA}(S, F)$ precisely captures (S, F)-algebras. Formally, we build a model transformation from $M \vDash \mathsf{ALGEBRA}(S, F)$ to an F-algebra $\alpha(M)$ and prove that α is essentially surjective (Theorem 6.5). A more precise analysis of the semantic equivalence between M and $\alpha(M)$ in terms of the set of F-equations that they validate is made in Section 6.3. The model transformation α is based directly on how matching logic models properly produce the interpretations of the inhabitant sets of sorts and operations, following Proposition 5.7.

Definition 6.2. Let (S, F) be a signature and $M \vDash \mathsf{ALGEBRA}(S, F)$. Then by Proposition 5.7, M produces the inhabitant set M_s for each sort $s \in S$ and the function $M_f : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ for each $f \in F_{s_1 \ldots s_n, s}$, restated as Corollary 6.3 for clarity. We define $A = \alpha(M)$ as the F-algebra where:

1. for $s \in S$, the carrier set $A_s = M_s$;
2. for each $f \in F_{s_1 \ldots s_n, s}$, the operation interpretation $A_f = M_f$.

Corollary 6.3. *Under the notations and conditions of Definition 6.2, we define $M_s = |[\![s]\!]|_M = inh_M \bar{\cdot} \{s_M\}$ as the inhabitant set of s for each $s \in S$. For any $f \in F_{s_1 \ldots s_n, s}$ and $x_1 \in M_{s_1}, \ldots, x_n \in M_{s_n}$, we know by axiom* (Function) *that $f_M \bar{\cdot} \langle x_1, \ldots, x_n \rangle_M$ is a singleton, whose (unique) element we denote $M_f(x_1, \ldots, x_n)$. Thus, M_f is a function from $M_{s_1} \times \cdots \times M_{s_n}$ to M_s, for each $f \in F_{s_1 \ldots s_n, s}$.*

Remark 6.4. Although the matching logic model M has a nonempty carrier set by definition, the derived algebra $\alpha(M)$ can have empty carrier sets. This is because the carrier set A_s of sort s is $M_s = |[\![s]\!]|_M$. Therefore, if $[\![s]\!]$ is interpreted by M as the empty set then A_s is also empty.

Next, we state the theorem that shows that α is essentially surjective.

Theorem 6.5. *Let (S, F) be a signature and A be any (S, F)-algebra. Then there exists a matching logic model $M \vDash \mathsf{Signature}(S, F)$ such that $\alpha(M)$ is exactly A.*

It turns out that the most technically tedious proof of all results in this paper is to show the existence of *any* model of $\mathsf{ALGEBRA}(S, F)$, that is, to show the consistency of the specification $\mathsf{ALGEBRA}(S, F)$. Indeed, $\mathsf{ALGEBRA}(S, F)$ imports $\mathsf{PRELUDE}$, which not only defines pairs and functions, but also pairs of pairs, functions of functions, and all possible nested combinations. Therefore, we exile the proof of Theorem 6.5, which is straightforward but tedious, to Appendix D.

6.3 Capturing Equational Specifications

We now study the semantic equivalence between M and the derived algebra $\alpha(M)$. We show that they validate the same F-equations (there are two validity relations: the matching logic validity in Definition 4.7 and the algebra validity in Definition 3.8), but first we define F-equations as patterns.

By Definition 3.7, an F-equation is associated with an S-sorted variable set V where each variable $x \in V$ is associated with its sort, denoted $\mathsf{sort}(x) \in S$. For technical convenience, we assume that all sorted variables used in F-equations are element variables of matching logic, that is, $V \subseteq EV$. Then, F-equations can be defined using matching logic equality and sorted quantification. Formally,

Definition 6.6. For an equational specification (F, E), we define $\mathsf{EQSPEC}(S, F, E)$ in Specification 12, abbreviated $\mathsf{EQSPEC}(F, E)$ or $\mathsf{EQSPEC}(E)$.

In $\mathsf{EQSPEC}(F, E)$, an F-equation $\forall V . t = t'$ for $t, t' \in T_F(V)$ is a well-formed pattern, because t and t' are application patterns (see $\mathsf{PRELUDE}$) and equality is defined in Section 4.3.1.

Since $\mathsf{EQSPEC}(F, E)$ imports $\mathsf{ALGEBRA}(F)$, we can apply the model transformation α on a model $M \vDash \mathsf{EQSPEC}(F, E)$, and the result, $\alpha(M)$, is an F-algebra by Definition 6.2. In the following, we show that $\alpha(M)$ is actually an (F, E)-algebra by proving the stronger result, that $\alpha(M)$ and M validate the same set of F-equations.

spec $\mathsf{EQSPEC}(S, F, E)$
 Imports: $\mathsf{ALGEBRA}(S, F)$
 Notations:
  // $V = \{x_1, \ldots, x_n\}$ is a set of sorted variables
  $\forall V . \varphi \equiv \forall x_1 : \mathsf{sort}(x_1) . \ldots \forall x_n : \mathsf{sort}(x_n) . \varphi$
  $\exists V . \varphi \equiv \exists x_1 : \mathsf{sort}(x_1) . \ldots \exists x_n : \mathsf{sort}(x_n) . \varphi$
 Axioms: (Equation) e for every $e \in E$
endspec

Specification 12: $\mathsf{EQSPEC}(S, F, E)$

Theorem 6.7. *Let $M \vDash \mathsf{EQSPEC}(F, E)$ and $\alpha(M)$ be the derived F-algebra. Then, for any F-equation e, we have $M \vDash e$ iff $\alpha(M) \vDash_{\mathsf{Alg}} e$. Particularly, we know that $\alpha(M)$ is an (F, E)-algebra.*

Remark 6.8. Theorem 6.7 together with Theorem 6.5 actually show that there exists a theoroidal comorphism $\mathbb{ALG}^{\mathsf{th}} \to \mathbb{ML}^{\mathsf{th}}$ from the category of many-sorted E-algebras to the category of matching logic $\mathsf{EQSPEC}(E)$-models; details in Appendix D.3.

Finally, we show that the specification $\mathsf{EQSPEC}(E)$ is a *conservative extension* of E:

Theorem 6.9. *For any equational specification E and any equation e, the following are equivalent: (1) $\mathsf{EQSPEC}(E) \vdash e$; (2) $\mathsf{EQSPEC}(E) \vDash e$; (3) $E \vDash_{\mathsf{Alg}} e$; (4) $E \vdash_{\mathsf{Alg}} e$.*

Proof. We prove (1) $\Longrightarrow$ (2) $\Longrightarrow$ (3) $\Longrightarrow$ (4) $\Longrightarrow$ (1). (1) $\Longrightarrow$ (2) is by the soundness of matching logic (Theorem 4.13). (2) $\Longrightarrow$ (3) is by the semantic equivalence (Theorem 6.7) and the surjectivity of α (Theorem 6.5). (3) $\Longrightarrow$ (4) is by the completeness of equational deduction (Theorem 3.11). (4) $\Longrightarrow$ (1) holds because the matching logic proof system supports equational reasoning (Proposition 4.12). $\square$

6.3.1 A Pitfall w.r.t. Theorem 6.9

Recall that the initial E-algebra I has no-confusion (Theorem 3.22), that is, $I \vDash_{\mathsf{Alg}} e$ iff $E \vdash_{\mathsf{Alg}} e$. Combing that with Theorem 6.9, one may conclude that all valid equations in the initial algebra I can be inferred using equational deduction, and that is all, since we now have a sound and complete solution to formal reasoning in initial algebra semantics.

The above conclusion is, of course, wrong, because after all, (equational) validity in initial algebras is Π^0_2-complete (see Section 9) and thus has no complete (and effective) proof system like equational deduction. The problem is that no-confusion of I only works on *ground equations*, not all equations. Therefore, equational deduction is only complete w.r.t. *ground* equational validity, but not equational validity in general. In fact, in Section 9, we will give a concrete example where equational deduction fails to prove a valid equation (with variables) of I, which can instead be proved using the matching logic proof system (Section 4.4). For that, we need to capture initiality by matching logic patterns/axioms and support induction—the main topics in Sections 7-9.

7 Capturing Term Algebras

In this section, we define the matching logic specifications $\mathsf{TERMALGEBRA}(F)$ that capture the term algebras (Definition 3.14), by taking the specification $\mathsf{ALGEBRA}(F)$ for F-algebras and define two additional axioms, (No Confusion) and (No Junk), that capture the no-confusion and no-junk properties (Theorem 3.22). We show that for any model $M \vDash \mathsf{TERMALGEBRA}(F)$, the derived algebra $\alpha(M)$ is exactly the term algebra T_F. Specifically, we will discuss no-confusion in Section 7.1 and no-junk in Section 7.2. Then we put them together in Section 7.3 and state the main theorem.

7.1 Capturing Term Algebras: No-Confusion

In Theorem 3.22, no-confusion and no-junk were defined for all (F, E)-algebras, in general; in this section we are only considering the special case when $E = \emptyset$, that is, when there are no underlying equations. In this special case, no-confusion takes a simpler form, which we state in Lemma 7.1.

Lemma 7.1. *Let $(F, \emptyset)$ be an equational specification with no equational axioms. Then, an $(F, \emptyset)$-algebra A satisfies no-confusion iff (1) A_f is injective for each $f \in F$, and (2) the ranges/codomains of A_f and $A_{f'}$ are disjoint for all distinct $f, f' \in F$.*

In other words, in term algebra T_F all operations in F are constructors; two terms are equal iff they are built from the same operation and the same argument(s). This leads us to the following:

Definition 7.2. For a signature (S, F), we define $\mathsf{NOCONFUSION}(S, F)$ in Specification 13, abbreviated $\mathsf{NOCONFUSION}(F)$.

Intuitively, (Distinct Function) states that distinct operations are indeed different functions. This is true in term algebras, where different operations build different terms, so by axiom (Func Ext) in FUNCTION, they are different functions. (No Confusion) captures no-confusion by one axiom, where sorts *SigOps* and *SigArgs* (Definition 6.1) restrict the ranges accordingly.

```
spec NOCONFUSION(S, F)
 Imports: ALGEBRA(S, F)
 Axioms:
 (Distinct Function)  f ≠ f'   for distinct f, f' ∈ F
 (No Confusion)       ∀f, f' : SigOps. ∀args, args' : SigArgs.
                        (f args)=(f' args')→f=f'∧args=args'
endspec
```

Specification 13: NOCONFUSION(S, F)

Lemma 7.3. *For any $M \vDash$ NOCONFUSION(F), $\alpha(M)$ satisfies no-confusion. Particularly, for any $t, t' \in T_F$, these are equivalent: (1) $M \vDash t = t'$; (2) $\alpha(M) \vDash_{\mathsf{Alg}} \forall\emptyset.\, t = t'$; (3) t and t' are the same.*

7.2 Capturing Term Algebras: No-Junk

By Theorem 3.22, an algebra A satisfies no-junk if, intuitively, its carrier sets are generated by the sets of ground terms; that is, they are the *smallest sets* that are closed under the operations in F. Therefore, no-junk can be defined axiomatically by the μ-binder and fixpoint patterns (Section 4.2.3).

Let us first build some intuition with a few simple examples.

Example 7.4. Let (S, F) be a signature where $S = \{s\}$ has one sort and $F = \{a \in F_{\epsilon,s}, f \in F_{s,s}, g \in F_{s\,s,s}\}$ has a constant a, a unary operation f, and a binary operation g. Then, we define no-junk as:

$$[\![s]\!] = \mu D.\, a \vee f(D) \vee g(D, D)$$

Intuitively, it specifies that $[\![s]\!]$ is the smallest set D that includes a and is closed under f and g.

Example 7.4 is in principle the same as axiom (Nat Domain) in specification NN that captures natural numbers; see Specification 3. Since the signature has only one sort, its carrier set only depends on itself. In recursive data types, this is called *single recursion* or *direct recursion*. In general, however, the signature (S, F) may include many sorts and also operations among them, which causes *mutual recursion*. We will follow the usual way to convert mutual recursion to single recursion, of which the main idea is shown in the following example.

Example 7.5. Let (S, F) be a signature where $S = \{s_1, s_2\}$ and $F = \{a_1 \in F_{\epsilon,s_1}, a_2 \in F_{\epsilon,s_2}, f \in F_{s_1\,s_2,s_2}, g \in F_{s_1\,s_2,s_1}\}$. Here, a_1, a_2 are two constants and f, g are two binary functions. The following is a failed attempt that uses the μ-binder to capture the mutual recursion between $[\![s_1]\!]$ and $[\![s_2]\!]$:

$$\langle [\![s_1]\!], [\![s_2]\!] \rangle = \mu\, \langle D1, D2 \rangle\, .\, \langle a_1 \vee g(D_1, D_2), a_2 \vee f(D_1, D_2) \rangle \qquad \text{// wrong use of } \mu$$

The above definition, although intuitive and straightforward, is wrong, because μ can only bind a set variable, and not a structure such as $\langle D_1, D_2 \rangle$. We can correct it by replacing $\langle D_1, D_2 \rangle$ with a set variable D and use the projection function to restore D_1 and D_2. The corrected definition is:

$$\langle [\![s_1]\!], [\![s_2]\!] \rangle = \mu D.\, \langle a_1 \vee g((proj\ 1\ D), (proj\ 2\ D)), a_2 \vee f((proj\ 1\ D), (proj\ 2\ D)) \rangle$$

where $(proj\ i\ D)$ is the projection of D, for $i \in \{1, 2\}$, defined in Specification 8.

Next example is about *void sorts*. Given a signature F, a sort s is *void* in F if it has no ground terms, i.e., $T_{F,s} = \emptyset$. The following example shows why void sorts require special treatment:

Example 7.6. Let (S, F) be a signature where $S = \{s_3, s_4\}$ and $F = \{b \in F_{\epsilon,s_3}\}$. Then s_3 is a non-void sort and s_4 is a void sort. If we follow Example 7.5 to define the carrier sets $[\![s_3]\!]$ and $[\![s_4]\!]$, we get:

$$\langle [\![s_3]\!], [\![s_4]\!] \rangle = \mu D.\, \langle b, \bot \rangle \qquad \text{// this is a wrong definition}$$

spec $\mathsf{NOJUNK}(S, F)$
Imports: $\mathsf{ALGEBRA}(S, F)$
Axioms:

(No Junk Void) $\llbracket s \rrbracket = \bot$ for each $s \in S_{\mathsf{void}}$

(No Junk Non-Void) $$\langle \llbracket s_1 \rrbracket, \ldots, \llbracket s_n \rrbracket \rangle = \mu D. \left\langle \bigvee_{f \in F_{s^1_1 \ldots s^1_{m_1}, s_1}} f\left(D_{s^1_1}, \ldots, D_{s^1_{m_1}}\right), \ldots, \bigvee_{f \in F_{s^n_1 \ldots s^n_{m_n}, s_n}} f\left(D_{s^n_1}, \ldots, D_{s^n_{m_n}}\right) \right\rangle$$

where $D_s \equiv (\mathit{proj}\ i\ D)$ if s is s_i for some $s_i \in S$, or $D_s \equiv \bot$ if $s \in S_{\mathsf{void}}$

endspec

Specification 14: $\mathsf{NOJUNK}(S, F)$

However, the above is wrong, because one can show that $\langle b, \bot \rangle$ equals to $\bot$, due to pointwise extension (Definition 4.4). In other words, an empty fragment $\bot$ empties the entire structure $\langle b, \bot \rangle$, which makes the fixpoint pattern $\mu D.\ \langle b, \bot \rangle$ also empty. Therefore, both s_3 and s_4 have an empty carrier set, which is clearly not intended. To avoid the void sorts propagating their emptiness to the non-void sorts, we separate them and define the void sorts first. Non-void sorts are then defined like in Example 7.5. This leads us to the following specification of no-junk.

Definition 7.7. Let (S, F) be a signature and $S = S_{\mathsf{void}} \cup S_{\mathsf{nonvoid}}$, where S_{void} includes void sorts and $S_{\mathsf{nonvoid}} = \{s_1, \ldots, s_n\}$ includes non-void sorts. We define $\mathsf{NOJUNK}(S, F)$ in Specification 14, abbreviated $\mathsf{NOJUNK}(F)$, where $s, s_i, s^j_i, \ldots$ are used to range over the sorts in S.

(No Junk Void) defines the carrier sets of void sorts to be empty. (No Junk Non-Void) defines the carrier sets of non-void sorts $\llbracket s_1 \rrbracket, \ldots, \llbracket s_n \rrbracket$ simultaneously, using one fixpoint pattern $\mu D.\ \langle \cdots \rangle$, in which the ith component $(1 \leq i \leq n)$ is a disjunction pattern over all operations whose return sort is s_i applied to the projections of D that correspond to the apporpriate argument sorts. For brevity, we refer to both (No Junk Void) and (No Junk Non-Void) as simply (No Junk).

Lemma 7.8. *For $M \vDash \mathsf{NOJUNK}(S, F)$, the derived algebra $A = \alpha(M)$ satisfies no-junk. Specifically, for any $s \in S$ and $a \in A_s$, there exists a ground term $t_a \in T_{F,s}$ such that $\mathsf{eval}(t_a) = a$ (see Remark 3.6).*

7.3 Capturing Term Algebras: No-Junk + No-Confusion

The specification $\mathsf{TERMALGEBRA}(S, F)$ of term algebras, abbreviated $\mathsf{TERMALGEBRA}(F)$, is obtained by simply importing $\mathsf{NOJUNK}(S, F)$ and $\mathsf{NOCONFUSION}(S, F)$.

Theorem 7.9. *For $M \vDash \mathsf{TERMALGEBRA}(F)$, the derived algebra $\alpha(M)$ is the term algebra T_F.*

Remark 7.10. [Malc'ev, 1936] shows a complete FOL axiomatization of term algebras based on the same no-junk and no-confusion characterizations; see also [Kovács et al., 2017]. Since FOL has no direct support for fixpoints or induction, the no-confusion part is actually weaker, and only requires all elements to be built from the operations (to see why it is weaker, consider real numbers, where each real number r is built from *succ* on r' for $r' = r - 1$). The FOL axiomatization is a complete theory, meaning that for any FOL sentence φ, either φ or $\neg\varphi$ can be proved. By the completeness of FOL deduction, it is decidable to determine whether FOL sentences are valid in term algebras.

This classic complete FOL axiomatization of term algebras is (understandably) weaker than our result. First, it does not precisely capture the term algebras in terms of *models*: indeed it allows arbitrarily large models due to the Löwenheim-Skolem theorem [Löwenheim, 1915]. Second, it is not extensible: it requires that all operations are constructors with no underlying axioms, so one cannot take the complete axiomatization of constructors *zero* and *succ*, and extend it with two defined functions *plus* and *mult* with their Peano axioms, and obtain a complete axiomatization of natural numbers with addition and multiplication—the completeness is lost during extension. In contrast, the matching logic specification $\mathsf{TERMALGEBRA}(F)$ captures precisely term algebras semantically, and its extensibility is what allows us to define equational axioms

spec INITIALALGEBRA(S, F, E)
Imports: TERMALGEBRA(S, F)
Symbols: $Eq, idRel, converseRel, composeRel, congRel$
Notations:
$\varphi_1 \subsetneq \varphi_2 \equiv \forall x_1 .\, x_1 \in \varphi_1 \to \exists x_2 .\, x_2 \in \varphi_2 .\ \langle x_1, x_2 \rangle \in Eq$
$\varphi_1 \simeq \varphi_2 \equiv \varphi_1 \subsetneq \varphi_2 \wedge \varphi_2 \subsetneq \varphi_1$
$R^{-1} \equiv converseRel\ R$
$R_1 \circ R_2 \equiv composeRel\ R_1\ R_2$
Axioms:
(Identity) $idRel = \bigvee_{s \in S} \exists x{:}s.\ \langle x, x \rangle$
(Converse) $R^{-1} = \exists x. \exists y.\ \langle y, x \rangle \wedge (\langle x, y \rangle \in R)$
(Composition) $R_1 \circ R_2 = \exists x. \exists y. \exists z.\ \langle x, z \rangle \wedge (\langle x, y \rangle \in R_1 \wedge \langle y, z \rangle \in R_2)$
(Congruence) $congRel\ R = \bigvee_{f \in F_{s_1 \ldots s_n, s}} \exists x_1, y_1{:}s_1 \ldots \exists x_n, y_n{:}s_n.\ \langle f(x_1, \ldots, x_n), f(y_1, \ldots, y_n) \rangle \wedge \bigwedge_{1 \le i \le n} \langle x_i, y_i \rangle \in R$
(Equivalence) $Eq = \mu R.\, idRel \vee R^{-1} \vee (R \circ R) \vee (congRel\ R) \vee \bigvee_{(\forall V.\, t = t') \in E} \exists V.\ \langle t, t' \rangle$
endspec

Specification 15: INITIALALGEBRA(S, F, E)

constraining the operations and to capture initial E-algebras for an equational specification E, discussed in Section 8.

8 Capturing Initial E-Algebras

Previously, we showed how to capture term algebras, which are initial E-algebras when $E = \emptyset$. In this section, we show how to capture initial E-algebras in general, axiomatically in matching logic. We take the specification TERMALGEBRA(F) that captures the term algebra T_F and define on top of it the congruence relation $\simeq_E$; see Section 3.4. The resulting specification, denoted INITIALALGEBRA(F, E), defines the term algebra T_F and the congruence relation $\simeq_E$ over terms, and thus captures the quotient term algebra $T_{F/E}$, i.e., the initial E-algebra (Theorem 3.20).

Let (S, F, E) be an equational specification. The congruence relation $\simeq_E$ is the *smallest relation* that includes the identity relation and all instances of the equations in E, and is closed under converse, composition, and congruence w.r.t. all operations in F. Therefore, it can be axiomatically defined by the μ-binder and fixpoint patterns. In matching logic, a binary relation R can be represented by a pattern that is matched by the pairs of all elements that are in relation R. This inspires the following definition, where we use a symbol Eq to capture the congruence relation[1] $\simeq_E$.

Definition 8.1. Let (S, F, E) be an equational specification. We define INITIALALGEBRA(S, F, E) in Specification 15, abbreviated INITIALALGEBRA(F, E) or INITIALALGEBRA(E).

INITIALALGEBRA(E) firstly defines several operations and notations for dealing with (binary) relations. Intuitively, *idRel* is the identity relation on the elements in the carrier sets of sorts in S. R^{-1} and $R_1 \circ R_2$ are the converse of R and composition of R_1 and R_2, respectively. *congRel* R is the relation obtained by propagating R through all the operations in F. The last axiom (Equivalence) defines Eq as the smallest relation R that includes *idRel*, is closed under converse, composition, and congruence, and includes all equations in E. Finally, we write $\varphi_1 \subsetneq \varphi_2$ to mean that φ_1 is included in φ_2 modulo the relation Eq, and $\varphi_1 \simeq \varphi_2$ to mean that φ_1 and φ_2 are the same modulo relation Eq.

The following theorem states that Eq indeed captures the congruence relation $\simeq_E$.

[1]Alternatively, we could have defined Eq as a sort inhabited by the pairs in $\simeq_E$ with constructors the equations in E, identities, converse, and composition. This would allow us to use Eq in sort constructors (pairs, tuples, functions) similarly to how identity types are used in higher inductive types [Kaposi et al., 2019; Fiore et al., 2020]. This is left as future work.

Theorem 8.2. *For any $M \vDash \mathsf{INITIALALGEBRA}(S, F, E)$, the derived model $\alpha(M)$ is exactly the term algebra T_F (Theorem 7.9). Then, the interpretation Eq_M is a binary relation over ground terms in T_F, and we have that $(t, t') \in Eq_M$ iff $t \simeq_E t'$ for all $t, t' \in T_F$, where $\simeq_E$ is defined in Proposition 3.15.*

Thus, Eq_M is exactly the congruence relation $\simeq_M$. This naturally leads us to the following theorem.

Theorem 8.3. *Under the conditions and notations in Theorem 8.2, we define $\beta(M)$ as the Eq_M-quotient algebra of $\alpha(M)$. Then, $\beta(M)$ is exactly the quotient term algebra $T_{F/E}$.*

Note that for a model $M \vDash \mathsf{INITIALALGEBRA}(F, E)$, we have defined two model transformations: α (Definition 6.2) and β (Theorem 8.3). $\alpha(M)$ gives us the term algebra T_F, where equations in E are ignored. $\beta(M)$ builds upon $\alpha(M)$ by taking the quotient algebra w.r.t. the congruence relation Eq_M. It is important to note that the equality $t = t'$ in M (recall that equality is defined in Specification 2) always means the true equality between the ground terms t and t', and *not* the $\simeq_E$-equivalence relation. One can see that clearly from Lemma 7.3, which states that $M \vDash t = t'$ iff t and t' are syntactically the same terms. To refer to the $\simeq_E$-equivalence relation, one should use $t \simeq t'$, which is defined as a notation in Specification 15. In fact, the following theorem shows that $t \simeq t'$ captures precisely the equality in the quotient term algebra $T_{F/E}$:

Theorem 8.4. *Let $M \vDash \mathsf{INITIALALGEBRA}(F, E)$ and $T_{F/E}$ be the quotient term algebra. Then for any ground or non-ground equations $\forall V.\, t = t'$, we have $T_{F/E} \vDash_{\mathsf{Alg}} \forall V.\, t = t'$ iff $M \vDash \forall V.\, t \simeq t'$.*

Proof. The proof follows directly from Theorems 8.2 and 8.3. Note that a non-ground equation holds iff it holds under all valuations of variables, which is then guaranteed by Theorem 8.2. □

Theorems 8.3 and 8.4 are important for various reasons. First, they show that the matching logic theory $\mathsf{INITIALALGEBRA}(E)$ captures precisely the initial E-algebra semantics, for any E. Second, they establish a semantic equivalence between validity in initial E-algebra and matching logic validity in $\mathsf{INITIALALGEBRA}(E)$. Third, they suggest that we can use the proof system of matching logic to prove, within specification $\mathsf{INITIALALGEBRA}(E)$, equations/patterns of the form $\forall V.\, t \simeq t'$. By the soundness of matching logic (Theorem 4.13) all proved equations are valid in initial E-algebras, that is, $\mathsf{INITIALALGEBRA}(E) \vdash \forall V.\, t \simeq t'$ implies $T_{F/E} \vDash_{\mathsf{Alg}} \forall V.\, t = t'$. Consequently, *matching logic provides a formal initial algebra reasoning framework.* We illustrate it in Section 9.

9 Inductive Reasoning in Initial Algebra using Matching Logic

Initial algebra semantics reasoning asks what truths about the initial (E-)algebras we can prove, and how. By Theorem 3.11, we know that all valid ground equations of an initial algebra can be recursively enumerated by the complete equational proof system (Definition 3.10), but not for non-ground equations, of which the problem is Π^0_2-complete [Subrahmanyam, 1990]. Hence, one cannot hope to have any automated procedure to prove and/or disprove all equations in initial algebras.

Initial algebra reasoning is (almost) a synonym for *induction*. The application of various inductive techniques in formal verification of programs flourished in the 1960s [Burstall, 1969; McCarthy, 1963; Cooper, 1966; Mccarthy and Painter, 1967; Burstall, 1968; Painter, 1967; Kaplan, 1967]. Later, it was discovered that initiality, or more precisely, the *no-junk* property (Theorem 3.22), is what powers induction and induction-based proof techniques in the initial algebra semantics [Meseguer and Goguen, 1985, Proposition 16]. Intuitively, an algebra A is no-junk means all elements in A can be represented by some terms, that is, the unique morphism $f_A : T_{F/E} \to A$ from the initial algebra (quotient term algebra) $T_{F/E}$ to A is surjective. Since $T_{F/E}$ is constructed inductively based on terms, it enjoys inductive reasoning, which can then be "mapped" to A through the unique morphism f_A, whose surjectivity guarantees that all elements in A are covered by the induction. Since then, various induction principles have been adopted as alternative equivalents of initiality; see, e.g., [Meseguer and Goguen, 1985, Section 4.4] and derived practical implementations and tools [Clavel et al., 2020].

In matching logic, no-junk is captured by one axiom (No Junk) using a fixpoint pattern (Section 7.2). The matching logic proof system also has one proof rule, (Knaster-Tarski) that is dedicated to fixpoint reasoning

(Section 4.4). In this section, we will show how induction in initial algebra reduces to matching logic proofs that make use of the proof rule (Knaster-Tarski) together with the axiom (No Junk), and inductive proof techniques become matching logic proof heuristics.

We illustrate the above by means of an example. Let us consider the set of natural numbers defined as the initial algebra of operations $\{zero, succ\}$, and then let us define an operation *plus* for addition, using the two (Peano) axioms $E^{Nat} = \{\forall x{:}Nat.\, plus(x, zero) \simeq x, \forall x, y{:}Nat.\, plus(x, succ(y)) \simeq succ(plus(x, y))\}$. Our goal is to prove that *zero* is also the left identity of *plus*, that is:

$$\forall y{:}Nat.\, plus(zero, y) \simeq y \tag{†}$$

Firstly, we point out a known fact that (†) does *not* hold in all algebras that validate the Peano axioms. As a counterexample, consider an algebra with only two elements $\{0, \star\}$, where *zero* is interpreted as 0, *succ* is interpreted as the identity function on $\{0, \star\}$, and *plus* is interpreted as the first projection (i.e., returns its first argument). The reader is encouraged to check that both equations in E^{Nat} hold, but Eq. (†) does not. The property holds, as expected, if we impose the additional initial algebra semantics requirement. However, by default initial algebra semantics makes no distinction between so-called "constructors" and "defined functions", treating them uniformly as operations/constructors. In practice, this results in unnecessarily tedious inductive proofs, where an induction case is needed for each operation, be it an intended constructor or not. It is the equations that make some operations "defined" in terms of the "constructors", which leads to simpler and more intuitive proofs. In our case here, a usual inductive proof of (†) using the initial algebra approach involves the following three steps: (1) prove that *plus* is *well-defined* on natural numbers constructed with *zero* and *succ*; (2) apply *structural induction* on (†) using only the constructors *zero* and *succ*; and (3) prove the generated sub-goals, respectively.

Step (1) requires us to show that all ground instances of $plus(x, y)$ equal to the terms that are built from *zero* and *succ* only, so the axioms of *plus* indeed cover all cases and does not effectively create new ground terms. A common technique for such proofs is to observe that the two axioms/equations of *plus*, when oriented from left to right, become *rewrite rules* that always reduce the size of the sub-terms whose top-level operation is *plus*. Thus, the rewriting process for any ground term with *plus* will terminate at a canonical representation without *plus*, and thus *plus* is indeed well-defined. This effective approach to proving the well-definedness of an operation w.r.t. some equations goes back to [Jouannaud and Kounalis, 1989], and developed and improved in [Meseguer, 2012; Hendrix et al., 2006; Hendrix and Meseguer, 2007; Rocha and Meseguer, 2010; Hendrix, 2008; Comon et al., 2007] among many others, and has been implemented as a key feature of Maude [Clavel et al., 2020].

Step (2) applies structural induction on the variable y in (†) and yields the following two cases:

$$plus(zero, zero) \simeq zero \tag{3}$$
$$\forall z{:}Nat.\, (plus(zero, z) \simeq z) \to (plus(zero, succ(z)) \simeq succ(z)) \tag{4}$$

where (3) is often called the base case and (4) the induction step, meaning that its premise is exactly the original proof goal (†) and the conclusion is (†) but is applied by another *succ*. Finally, Step (3) simply proves (3) and (4) by standard equational reasoning, and we omit the details here.

The main inductive steps in the above proof are Steps (1) and (2), and there, the key is to show that *zero* and *succ* are the real and only constructors of natural numbers while *plus* is a (well-defined) function on the natural numbers. Without this observation, the initial algebra semantics induction principle in Step (2) will generate three cases, with an additional case for *plus*, making the proof much more complicated and challenging, as illustrated in [Comon, 2001, Section 2.4]. In practice, tools for inductive theorem proving using equational specifications have built-in support for users to declare certain operations as constructors and the rest as defined functions, following one (or both) of two aesthetically different but ultimately equivalent approaches:

1. A specification declares a sub-signature of constructors; Maude [Clavel et al., 2020] and proof assistants such as Coq [Coq Team, 2020] support this approach.

2. A sub-specification including only constructors is defined and then imported in a "protected" mode by a module system to the larger specification; OBJ [Goguen et al., 2000], CafeOBJ [Diaconescu and Futatsugi, 1998], and Maude [Clavel et al., 2020] support this approach.

In both cases, initiality is defined only for constructors and well-definedness needs to be proved for all defined functions. Both cases are *extensions* to the vanilla equational specifications that we defined in Section 3, where (1) adds constructor signatures and (2) adds a module system.

What is *not* an extension is the following axiomatic methodology offered by matching logic, where the statement that $\{zero, succ\}$ forms a constructor set can be expressed by a pattern/predicate and formally proved as a *theorem* of the matching logic specification $\mathsf{INITIALALGEBRA}(E^{Nat})$:

Theorem 9.1. $\mathsf{INITIALALGEBRA}(E^{Nat}) \vdash \underbrace{(\mu D.\, zero \vee succ(D) \vee plus(D, D))}_{equals\ to\ [\![Nat]\!]\ by\ axiom\ \text{(No Junk)}} \simeq (\mu D.\, zero \vee succ(D))$.

Intuitively, it states that the smallest set generated by $\{zero, succ, plus\}$ is indeed the same as the smallest set generated by $\{zero, succ\}$, modulo the axioms in E^{Nat}. We then know that $\{zero, succ\}$ forms a constructor set *and* that *plus* is well-defined, precisely because it adds no new terms to the sort *Nat*. Therefore, we accomplished Step (1) in the above-mentioned common inductive proof using *only* matching logic reasoning. We emphasize that Theorem 9.1 is proved *within* the logic as an ordinary pattern at the object-level, using the proof system in Section 4.4, which requires no reasoning *outside* the formal system. This is in sharp contrast to the classical initial algebra semantics approaches to select an induction basis, such as [Jouannaud and Kounalis, 1989].

Next, we illustrate how to do the initial algebra inductive reasoning described above using fixpoint patterns and the (Knaster-Tarski) rule of matching logic. The idea is to derive proof goal Eq. (†) to an equivalent form where the LHS is a fixpoint pattern so we can apply (Knaster-Tarski). Specifically, let $\Psi \equiv \exists y{:}Nat.\, y \wedge (plus(zero, y) \simeq y)$ be the pattern matched by the natural numbers y that satisfy Eq. (†), that is, $plus(zero, y) \simeq y$. Then, we have the following reasoning steps:

$$\begin{array}{ll}
\mathsf{INITIALALGEBRA}(E^{Nat}) \vdash \forall y{:}Nat.\, plus(zero, y) \simeq y & \text{iff} \\
\mathsf{INITIALALGEBRA}(E^{Nat}) \vdash [\![Nat]\!] \to \Psi & \text{iff} \\
\mathsf{INITIALALGEBRA}(E^{Nat}) \vdash (\mu D.\, zero \vee succ(D)) \to \Psi & \text{if} \\
\mathsf{INITIALALGEBRA}(E^{Nat}) \vdash zero \to \Psi \text{ and } \mathsf{INITIALALGEBRA}(E^{Nat}) \vdash succ(\Psi) \to \Psi & \text{iff} \\
\mathsf{INITIALALGEBRA}(E^{Nat}) \vdash plus(zero, zero) \simeq zero & \\
\quad \text{and} \quad \mathsf{INITIALALGEBRA}(E^{Nat}) \vdash \forall y{:}Nat.\, (plus(zero, y) \simeq y) \to (plus(zero, succ(y)) \simeq succ(y)) &
\end{array}$$

which then restore the usual structural induction as shown in Eqs. (3)-(4).

In conclusion, the matching logic specification $\mathsf{INITIALALGEBRA}(E)$ not only precisely defines the initial E-algebra semantics (Theorem 8.3), but also yields inductive reasoning in it. The matching logic proof system in Section 4.4 only includes the most basic fixpoint rules based on (Knaster-Tarski), and yet it provides support to do *structural induction* and also *constructor analysis*, together with the axioms in $\mathsf{INITIALALGEBRA}(E)$. Therefore, our proposed specification $\mathsf{INITIALALGEBRA}(E)$ is interesting in terms of both models/semantics and formal (inductive) reasoning.

10 Extensions

So far, we have discussed how the standard, many-sorted initial algebra semantics is captured by matching logic. In this section, we apply the same methodology to various extensions. We do not have space to consider all variants of initial algebra listed in Section 3.5, so we only focus on three of them which are known to be both practical and challenging: (1) parametric specifications, where signatures and axioms are parameterized and can have many instances (e.g., parametric lists $PList\langle s\rangle$); (2) order-sorted algebras (OSA), where sorts are associated with an additional subsorting relation that enforces the corresponding inclusion relations between carrier sets (e.g., *Nat* is a subsort of *Int*); and (3) simultaneous inductive-recursive definitions, where the carrier sets of sorts and their constructors are inductively defined at the same time. We will discuss these extensions using examples. The reader will notice how *simple* and *natural* the resulting matching logic specifications are. We use these examples to make the point that matching logic provides a simple, powerful, and extensible framework for defining and reasoning about initial algebra semantics.

10.1 Parametric Equational Specifications

Here we discuss specifications with (sort) *parameters*. For example, the specification of *parametric lists* is $(\{s, \mathit{PList}\}, \{\mathit{nil} \in F_{\epsilon,\mathit{PList}}, \mathit{cons} \in F_{s\ \mathit{PList},\mathit{PList}}\}, \emptyset)$, where s is a *sort parameter*, *PList* constructs the sort of lists over base sort s, *nil* and *cons* are the constructors of *PList*, and there are no axioms. Parameter s can be instantiated to any sort. Therefore, this parametric specification is a "recipe" telling how to take an instance of parameter s and turn it into an instance of lists over s. In general, parameters need not be restricted to sorts (can include operations or even a whole specification).

Formally, a parametric specification is a *specification morphism* $\Phi\colon (\mathit{SP}, \mathit{FP}, \mathit{EP}) \to (S, F, E)$ from a *parameter* specification $(\mathit{SP}, \mathit{FP}, \mathit{EP})$ to a *parameterized* specification (S, F, E). Initial algebra semantics of Φ is defined using category theory, by considering the reduct functor $|_\Phi\colon \mathbf{Alg}(S, F, E) \to \mathbf{Alg}(\mathit{SP}, \mathit{FP}, \mathit{EP})$, whose left adjoint functor is associated with its free extension $(\mathcal{F}(A), \eta_A\colon A \to \mathcal{F}(A)|_\Phi)$ for each $A \in |\mathbf{Alg}(\mathit{SP}, \mathit{VP}, \mathit{EP})|$, such that for each $B \in |\mathbf{Alg}(S, F, E)|$ and $\varrho : A \to B|_\Phi$, there exists a unique morphism $\bar{\varrho} : \mathcal{F}(A) \to B$ with $\bar{\varrho}|_\Phi \circ \eta_A = \varrho$, and the initial semantics of Φ is given by the algebras $\mathcal{F}(A)$. Thus, for parametric lists, the parameter specification $(\{s\}, \emptyset, \emptyset)$ has only one sort parameter s and the parameterized specification is the one given at the beginning; the specification morphism is the inclusion, and the initial semantics is that of lists freely generated by the set interpreting the sort s; see, e.g., [Bergstra and Klop, 1983; Ehrig et al., 1984].

spec PLIST
Imports: PRELUDE
Symbols: $\mathit{PList}, \mathit{nil}, \mathit{cons}$
Notations:
$\mathit{PList}\langle s\rangle \equiv \mathit{PList}\ s$
$\mathit{nil}\langle s\rangle \equiv \mathit{nil}\ s$
$\mathit{cons}\langle s\rangle \equiv \mathit{cons}\ s$
Axioms: // all axioms are quantified by "$\forall s{:}\mathit{Sort}$"
(PList Sort) $\mathit{PList}\langle s\rangle : \mathit{Sort}$
(PList Nil) $\mathit{nil}\langle s\rangle : \mathit{PList}\langle s\rangle$
(PList Cons) $\mathit{cons}\langle s\rangle : s \otimes \mathit{PList}\langle s\rangle \ominus \mathit{PList}\langle s\rangle$
(No Junk) $[\![\mathit{PList}\langle s\rangle]\!] = \mu L.\, \mathit{nil}\langle s\rangle \vee \mathit{cons}\langle s\rangle([\![s]\!], L)$
(No Confusion) $\forall x{:}s.\, \forall l{:}\mathit{PList}\langle s\rangle.\, \mathit{nil}\langle s\rangle \neq \mathit{cons}\langle s\rangle(x, l)$
(No Confusion) $\forall x_1, x_2{:}s.\, \forall l_1, l_2{:}\mathit{PList}\langle s\rangle.$
$\mathit{cons}\langle s\rangle(x_1, l_1) = \mathit{cons}\langle s\rangle(x_2, l_2) \to x_1 = x_2 \wedge l_1 = l_2$
endspec

Specification 16: PLIST

While making admittedly elegant use of category theory and categorical concepts, the folklore approach towards parametric specifications is in our view still too complex and likely demotivating for practitioners. In addition, inductive reasoning is not handled in the core logic, but from the outside, at the categorical meta-level. After all technical preparation and setup, all is saying is that for any sort s, we have a *List* sort with constructors *nil* and *cons*, all parametric in s. This can be directly defined in matching logic in Specification 16, PLIST. We encourage the reader to compare PLIST with TERMALGEBRA(F), the specification of (non-parametric) term algebras in Section 7. The only difference is that PLIST allows axioms to be quantified/parametric by sorts.

10.2 Order-Sorted Algebras

Order-sorted algebra (OSA) extends many-sorted algebra by *subsorting* and *operation overloading*; e.g., we can define sort *Nat* to be a subsort of *Int*, written $\mathit{Nat} \leq \mathit{Int}$, and define the operation $\mathit{plus} \in F_{\mathit{Nat}\ \mathit{Nat},\mathit{Nat}} \cap F_{\mathit{Int}\ \mathit{Int},\mathit{Int}}$ as the (overloaded) addition on natural and integer numbers. OSA has many variants; [Goguen and Diaconescu, 1994] surveyed 13 different variants. Generally speaking, many-sorted initial algebra semantics can be extended to the order-sorted setting, but subtle technical conditions must be studied carefully, which are necessary to make the results listed in Section 3 also hold for OSA. We cannot discuss all technical details, but show at a high level how the two core extensions, subsorting and overloading, can be directly axiomatized in matching logic.

Formally, subsorting is a partial ordering on sorts. If $s \leq s'$, then it is required that in any algebras, the carrier sets $A_s \subseteq A_{s'}$. Therefore, to capture subsorting, we define the following axiom:

$$(\text{Subsorting}) \qquad [\![s]\!] \subseteq [\![s']\!]$$

for every $s \leq s'$, which states that the carrier set of s is included by the carrier set of s', as intended (set inclusion $\subseteq$ is defined as a notation in Specification 2). For operation overloading, let us consider an overloaded operation $f \in F_{s_1 \ldots s_n, s} \cap F_{s'_1 \ldots s'_n, s'}$. Thus, f can be applied to arguments of sorts $s_1, \ldots, s_n$, and of sorts $s'_1, \ldots, s'_n$, so we define one (Function) axiom in Section 5.4 for every arity of f (i.e., we use one matching logic symbol f for all overloaded copies/instances of operation f):

$$(\text{Function.1}) \qquad f : s_1 \otimes \cdots \otimes s_n \ominus s \qquad\qquad (\text{Function.2}) \qquad f : s'_1 \otimes \cdots \otimes s'_n \ominus s'$$

By extending the specification $\mathsf{TERMALGEBRA}(F)$ of (many-sorted) term algebras in Section 7 with the above two features, subsorting and overloading, we obtain the matching logic specification that captures order-sorted term algebras. In Appendix H, we pick one typical OSA variant that is proposed in [Goguen and Meseguer, 1992] as an example and work out the technical details.

10.3 Simultaneous Inductive-Recursive Definitions

Simultaneous inductive-recursive definitions consist of inductive sets and recursive functions that are inductively defined at the same time. Such definitions are often studied for *dependent type systems* [Cardelli, 1996; Martin-Löf, 1975], where the boundary between operations and sort/type constructors begins to blur. The following is a typical example[2] of simultaneous inductive-recursive definitions called *distinct-element lists*. It defines a sort *DList* of natural numbers built from *nil* and *cons*, but *cons* is mutually depended on a predicate $\mathit{fresh}(x, l)$ that states that number x is not in list l. By mutual dependence, we mean (1) *cons* will call *fresh* to check whether x is in l before it builds the extended list $\mathit{cons}(x, l)$, and (2) *fresh* is inductively defined on the lists of *DList* built from *cons*.

As we show below, the technique that we use to axiomatically define *DList* in matching logic is the same as the one we used to capture *mutual recursion* in Section 7.2. We can think of *fresh* as the "constructor" of a set *Fresh* that includes the pairs $\langle x, l \rangle$ such that $\mathit{fresh}(x, l)$ holds. Then, the two inhabitant sets $[\![\mathit{DList}]\!]$ and $[\![\mathit{Fresh}]\!]$ form a mutual recursion and can be axiomatically defined by:

$$\begin{aligned}
&\langle [\![\mathit{DList}]\!], [\![\mathit{Fresh}]\!] \rangle = \mu D.\ \langle \Phi_{\mathit{DList}}, \Phi_{\mathit{Fresh}} \rangle \\
&\qquad \text{where} \quad \Phi_{\mathit{DList}} \equiv \mathit{nil} \vee \exists x{:}\mathit{Nat}.\, \exists l.\ \mathit{cons}(x, l) \wedge l \in D_{\mathit{DList}} \wedge \langle x, l \rangle \in D_{\mathit{Fresh}} \\
&\qquad \phantom{\text{where}} \quad \Phi_{\mathit{Fresh}} \equiv \langle \mathit{nil}, [\![\mathit{Nat}]\!] \rangle \vee \exists x, y{:}\mathit{Nat}.\, \exists l.\ \langle x, \mathit{cons}(y, l) \rangle \wedge x \neq y \wedge \langle y, l \rangle \in D_{\mathit{Fresh}} \\
&\qquad \phantom{\text{where}} \quad D_{\mathit{DList}} \equiv \mathit{proj}\ 1\ D \\
&\qquad \phantom{\text{where}} \quad D_{\mathit{Fresh}} \equiv \mathit{proj}\ 2\ D
\end{aligned}$$

The same technique applies to all simultaneous inductive-recursive definitions we are aware of.

11 Conclusion

We showed that *initial algebra semantics* can be axiomatized in matching logic: given an equational specification E, we defined a corresponding matching logic specification $\mathsf{INITIALALGEBRA}(E)$ that captures precisely the initial E-algebras. Then, we showed how to use it to do inductive reasoning in *any* E using the *fixed* matching logic proof system. We discussed the many-sorted setting in detail, and covered three typical extensions—parametric specifications, order-sorted specifications, and simultaneous inductive-recursive definitions. In conclusion, we demonstrated that initial E-algebra semantics can be faithfully represented in matching logic, both theoretically and practically.

[2]This (artificial) example is described in [Dybjer, 2000, pp. 4], whose author gave the credit to Catarina Coquand.

References

Andreas Abel, Thierry Coquand, and Peter Dybjer. On the algebraic foundation of proof assistants for intuitionistic type theory. In Jacques Garrigue and Manuel V. Hermenegildo, editors, *Functional and Logic Programming*, pages 3–13, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-78969-7.

Egidio Astesiano, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella, and Andrzej Tarlecki. CASL: the common algebraic specification language. *Journal of Theoretical Computer Science*, 286(2):153–196, 2002. ISSN 0304-3975. doi: https://doi.org/10.1016/S0304-3975(01)00368-1. URL `http://www.sciencedirect.com/science/article/pii/S0304397501003681`. Current trends in Algebraic Development Techniques.

Steve Awodey, Nicola Gambino, and Kristina Sojakova. Inductive types in homotopy type theory. In *Proceedings ot the 27th Annual IEEE Symposium on Logic in Computer Science (LICS'12)*, pages 95–104, Dubrovnik, Croatia, 2012. IEEE.

Steve Awodey, Nicola Gambino, and Kristina Sojakova. Homotopy-initial algebras in type theory. *Journal of the ACM*, 63(6), January 2017. ISSN 0004-5411. doi: 10.1145/3006383. URL `https://doi.org/10.1145/3006383`.

Jan A. Bergstra and Jan Willem Klop. Initial algebra specifications for parametrized data types. *J. Inf. Process. Cybern.*, 19(1/2):17–31, 1983.

Denis Bogdănaş and Grigore Roşu. K-Java: A complete semantics of Java. In *Proceedings of the 42nd Symposium on Principles of Programming Languages (POPL'15)*, pages 445–456, Mumbai, India, 2015. ACM.

R. M. Burstall. Semantics of assignment. *Machine Intelligence*, 2:3–20, 1968.

R. M. Burstall. Proving properties of programs by structural induction. *The Computer Journal*, 12(1):41–48, 1969. ISSN 0010-4620. doi: 10.1093/comjnl/12.1.41.

R. M. Burstall and J. A. Goguen. *Algebras, theories and freeness: an introduction for computer scientists*, volume 91 of *NATO Advanced Study Institutes Series (Series C — Mathematical and Physical Sciences)*, chapter 11, pages 329–349. Springer, Dordrecht, Netherlands, 1982. doi: 10.1007/978-94-009-7893-5_11. URL `https://doi.org/10.1007/978-94-009-7893-5_11`.

Venanzio Capretta. Universal algebra in type theory. In Yves Bertot, Gilles Dowek, Laurent Théry, André Hirschowitz, and Christine Paulin, editors, *Theorem Proving in Higher Order Logics*, pages 131–148, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48256-7.

Luca Cardelli. Type systems. *ACM Computing Surveys (CSUR)*, 28(1):263–264, 1996.

Xiaohong Chen and Grigore Roşu. Matching μ-logic. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'19)*, pages 1–13, Vancouver, Canada, 2019a. IEEE. doi: 10.1109/LICS.2019.8785675.

Xiaohong Chen and Grigore Roşu. Matching μ-logic. Technical report, University of Illinois at Urbana-Champaign, 2019b. URL `http://hdl.handle.net/2142/102281`.

Xiaohong Chen and Grigore Roşu. A general approach to define binders using matching logic. In *Proceedings of the 25th ACM SIGPLAN International Conference on Functional Programming (ICFP'20)*, pages 1–32, New Jersey, USA, 2020a. ACM. URL `http://hdl.handle.net/2142/106608`.

Xiaohong Chen and Grigore Roşu. A general approach to define binders using matching logic. Technical report, University of Illinois at Urbana-Champaign, 2020b. URL `http://hdl.handle.net/2142/106608`.

Alonzo Church. *The calculi of lambda-conversion.* Princeton University Press, Princeton, New Jersey, USA, 1941. doi: 10.2307/2267126.

Manuel Clavel, Francisco Durán, Steven Eker, Santiago Escobar, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Carolyn Talcott. *Maude manual (version 3.0).* SRI International, 2020. URL `http://maude.lcc.uma.es/maude30-manual-html/maude-manual.html`.

H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. Release October 12th, 2007.

Hubert Comon. Inductionless induction. In Alan Robinson and Andrei Voronkov, editors, *Handbook of automated reasoning*, chapter 14, pages 913–962. North Holland, Amsterdam, 2001. doi: 10.1016/B978-044450813-3/50016-3.

D. C. Cooper. The equivalence of certain computations. *The Computer Journal*, 9(1):45–52, May 1966. ISSN 0010-4620. doi: 10.1093/comjnl/9.1.45.

Coq Team. Coq documents: calculus of inductive constructions. Online at `https://coq.inria.fr/refman/language/cic.html`., 2020.

Thierry Coquand and Christine Paulin. Inductively defined types. In *Proceedings of International Conference on Computer Logic*, pages 50–66, Tallinn, USSR, 1990. Springer Berlin Heidelberg. ISBN 978-3-540-46963-6. doi: 10.1007/3-540-52335-9_47.

Sandeep Dasgupta, Daejun Park, Theodoros Kasampalis, Vikram S. Adve, and Grigore Roşu. A complete formal semantics of x86-64 user-level instruction set architecture. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'19)*, pages 1133–1148, Phoenix, Arizona, USA, 2019. ACM.

Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Proceedings of the 25th International Conference on Automated Deduction Automated Deduction (CADE'15)*, pages 378–388, Cham, 2015. Springer International Publishing. ISBN 978-3-319-21401-6.

Razvan Diaconescu. *Institution-independent model theory.* Birkhäuser Basel, Boston, Berlin, 2008. ISBN 3764387076.

Razvan Diaconescu and Kokichi Futatsugi. *CafeOBJ report: the language, proof techniques, and methodologies for object-oriented algebraic specification*, volume 6 of *AMAST Series in Computing.* World Scientific, Singapore, 1998. doi: 10.1142/3831.

Peter Dybjer. Representing inductively defined sets by wellorderings in Martin-Löf's type theory. *Theoretical Computer Science*, 176(1):329–335, 1997. ISSN 0304-3975. doi: https://doi.org/10.1016/S0304-3975(96)00145-4. URL `http://www.sciencedirect.com/science/article/pii/S0304397596001454`.

Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *J. Symb. Log.*, 65(2):525–549, 2000. doi: 10.2307/2586554. URL `https://doi.org/10.2307/2586554`.

Hartmut Ehrig and Bernd Mahr. *Fundamentals of algebraic specification 1: equations and initial semantics*, volume 6 of *Monographs in Theoretical Computer Science. An EATCS Series.* Springer, Berlin Heidelberg, Germany, 1985. doi: 10.1007/978-3-642-69962-7.

Hartmut Ehrig, Hans-Jörg Kreowski, James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. Parameter passing in algebraic specification languages. *Theor. Comput. Sci.*, 28:45–81, 1984. doi: 10.1016/0304-3975(83)90065-8. URL `https://doi.org/10.1016/0304-3975(83)90065-8`.

Herbert B. Enderton. *A mathematical introduction to logic.* Academic Press, Califonia, USA, 1972. ISBN 978-0-12-238450-9.

M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 193–202, Trento, Italy, 1999. IEEE.

Marcelo P. Fiore and Chung-Kil Hur. On the construction of free algebras for equational systems. *Theor. Comput. Sci.*, 410(18):1704–1729, 2009. doi: 10.1016/j.tcs.2008.12.052. URL `https://doi.org/10.1016/j.tcs.2008.12.052`.

Marcelo P. Fiore, Andrew M. Pitts, and S. C. Steenkamp. Constructing infinitary quotient-inductive types. In Jean Goubault-Larrecq and Barbara König, editors, *Proceedings of the 23rd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'20) Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS'20)*, volume 12077 of *Lecture Notes in Computer Science*, pages 257–276, Dublin, Ireland, 2020. Springer. doi: 10.1007/978-3-030-45231-5_14. URL `https://doi.org/10.1007/978-3-030-45231-5_14`.

François Garillot, Georges Gonthier, Assia Mahboubi, and Laurence Rideau. Packaging mathematical structures. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics*, pages 327–342, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-03359-9.

Herman Geuvers, Randy Pollack, Freek Wiedijk, and Jan Zwanenburg. A constructive algebraic hierarchy in Coq. *J. Symb. Comput.*, 34(4):271–286, October 2002. ISSN 0747-7171. doi: 10.1006/jsco.2002.0552. URL `https://doi.org/10.1006/jsco.2002.0552`.

Joseph Goguen and Răzvan Diaconescu. An Oxford survey of order sorted algebra. *Mathematical Structures in Computer Science*, 4(3):363–392, 1994. doi: 10.1017/S0960129500000517.

Joseph Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, 2000. ISSN 0304-3975. doi: https://doi.org/10.1016/S0304-3975(99)00275-3. URL `http://www.sciencedirect.com/science/article/pii/S0304397599002753`.

Joseph Goguen and José Meseguer. Order-sorted algebra, part I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992. doi: 10.1016/0304-3975(92)90302-V.

Joseph Goguen, James Thatcher, Eric Wagner, and Jesse Wright. Initial algebra semantics and continuous algebras. *Journal of the ACM*, 24(1):68–95, 1977.

Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. *Software engineering with OBJ: Algebraic specification in action*, chapter Introducing OBJ, pages 3–167. Springer, Massachusetts, USA, 2000.

Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992. URL `http://doi.acm.org.proxy2.library.illinois.edu/10.1145/147508.147524`.

Joseph A. Goguen and Grant Malcolm. *Algebraic semantics of imperative programs.* MIT Press, Cambridge, MA, USA, 1996. ISBN 026207172X.

Joseph A. Goguen and José Meseguer. Completeness of many-sorted equational logic. *Houston Journal of Mathematics*, 11(3):307–334, 1985.

Joseph A. Goguen and Grigore Rosu. Institution morphisms. *Formal Asp. Comput.*, 13(3-5):274–307, 2002. doi: 10.1007/s001650200013. URL `https://doi.org/10.1007/s001650200013`.

Jan Friso Groote and Radu Mateescu. Verification of temporal properties of processes in a setting with data. In Armando M. Haeberer, editor, *Algebraic Methodology and Software Technology*, pages 74–90, Berlin, Heidelberg, 1999. Springer. ISBN 978-3-540-49253-5. doi: 10.1007/3-540-49253-4_8.

Emmanuel Gunther, Alejandro Gadea, and Miguel Pagano. Formalization of universal algebra in Agda. *Electronic Notes in Theoretical Computer Science*, 338:147–166, 2018. ISSN 1571-0661. doi: https://doi.org/10.1016/j.entcs.2018.10.010. URL `http://www.sciencedirect.com/science/article/pii/S1571066118300768`. The 12th Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2017).

J. V. Guttag and J. J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10(1): 27–52, March 1978. ISSN 1432-0525. doi: 10.1007/BF00260922.

John V. Guttag, James J. Horning, and Jeannette M. Wing. The larch family of specification languages. *IEEE Softw.*, 2(5):24–36, 1985. doi: 10.1109/MS.1985.231756. URL `https://doi.org/10.1109/MS.1985.231756`.

Alan G. Hamilton. *Logic for mathematicians.* Cambridge University Press, Cambridge, UK, 1978.

Chris Hathhorn, Chucky Ellison, and Grigore Roşu. Defining the undefinedness of C. In *Proceedings of the 36th annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15)*, pages 336–345, Portland, OR, 2015. ACM.

Joe Hendrix and José Meseguer. On the completeness of context-sensitive order-sorted specifications. In Franz Baader, editor, *Term Rewriting and Applications*, pages 229–245, Berlin, Heidelberg, 2007. Springer. ISBN 978-3-540-73449-9. doi: 10.1007/978-3-540-73449-9_18.

Joe Hendrix, José Meseguer, and Hitoshi Ohsaki. A sufficient completeness checker for linear order-sorted specifications modulo axioms. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, pages 151–155, Berlin, Heidelberg, 2006. Springer. ISBN 978-3-540-37188-5. doi: 10.1007/11814771_14.

Joseph D. Hendrix. *Decision procedures for equationally based reasoning.* PhD thesis, University of Illinois at Urbana-Champaign, 2008.

Everett Hildenbrandt, Manasvi Saxena, Xiaoran Zhu, Nishant Rodrigues, Philip Daian, Dwight Guth, Brandon Moore, Yi Zhang, Daejun Park, Andrei Ştefănescu, and Grigore Roşu. KEVM: A complete semantics of the Ethereum virtual machine. In *Proceedings of the 2018 IEEE Computer Security Foundations Symposium (CSF'18)*, pages 204–217, Oxford, UK, 2018. IEEE. `http://jellopaper.org`.

C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10): 576–580, 1969.

George Edward Hughes and Max Cresswell. *An introduction to modal logic.* Routledge, England, UK, 1968.

Patricia Johann and Neil Ghani. Initial algebra semantics is enough! In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications*, pages 207–222, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-73228-0.

Jean-Pierre Jouannaud and Emmanuel Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82(1):1–33, 1989. ISSN 0890-5401. doi: 10.1016/0890-5401(89)90062-X.

K Team. Matching logic proof checker. GitHub page `https://github.com/kframework/matching-logic-prover/tree/master/checker`, 2020.

D. M. Kaplan. Correctness of a compiler for Algol-like programs. *Stanford Artificial Intelligence Memo No. 48*, 48(1):1–35, 1967.

Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL), January 2019. doi: 10.1145/3290315. URL `https://doi.org/10.1145/3290315`.

Laura Kovács, Simon Robillard, and Andrei Voronkov. Coming to terms with quantified reasoning. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'17)*, pages 260–270, Paris, France, 2017. ACM. ISBN 978-1-4503-4660-3.

Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983. doi: 10.1016/0304-3975(82)90125-6.

B. Kutzler and F. Lichtenberger. *Bibliography on abstract data types*, volume 68 of *Informatik-Fachberichte*. Springer, New York, USA, 1983. doi: 10.1007/978-3-642-69032-7.

Leopold Löwenheim. Über möglichkeiten im relativkalkül. *Mathematische Annalen*, 76(4):447–470, 1915.

Anatoli Ivanovi Malc'ev. Axiomatizable classes of locally free algebras of various type. *The Metamathematics of Algebraic Systems: Collected Papers*, 1(1):262–281, 1936.

Vincenzo Manca and Antonino Salibra. Soundness and completeness of the Birkhoff equational calculus for many-sorted algebras with possibly empty carrier sets. *Theoretical Computer Science*, 94(1):101–124, 1992. ISSN 0304-3975. doi: 10.1016/0304-3975(92)90325-A.

Per Martin-Löf. An intuitionistic theory of types: predicative part. In *Logic Colloquium*, volume 80, pages 73–118. Elsevier, Amsterdam, The Netherlands, 1975.

Coq Team. *The Coq proof assistant.* LogiCal Project, 2020. URL `http://coq.inria.fr`.

John McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 33–70. Elsevier, Amsterdam, The Netherlands, 1963. doi: 10.1016/S0049-237X(08)72018-4.

John Mccarthy and James Painter. Correctness of a compiler for arithmetic expressions. In *Proceedings of Symposiain Applied Mathematics*, volume 19, pages 33–41, Rhode Island, USA, 1967. American Mathematical Society.

José Meseguer. Conditioned rewriting logic as a united model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992. doi: 10.1016/0304-3975(92)90182-F. URL `https://doi.org/10.1016/0304-3975(92)90182-F`.

José Meseguer. Membership algebra as a logical framework for equational specification. In Francesco Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques (WADT'97)*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61, Tarquinia, Italy, 1997. Springer. doi: 10.1007/3-540-64299-4_26. URL `https://doi.org/10.1007/3-540-64299-4_26`.

José Meseguer. Twenty years of rewriting logic. *The Journal of Logic and Algebraic Programming*, 81(7–8):721–781, 2012. doi: 10.1016/j.jlap.2012.06.003.

José Meseguer and Joseph A. Goguen. Initiality, induction, and computability. In *Algebraic Methods in Semantics*, pages 459–543. Cambridge University Press, New York, USA, 1985.

Ulf Norell. *Towards a practical programming language based on dependent type theory.* PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.

Ulf Norell. Dependently typed programming in Agda. In *Proceedings of the 6th International Conference on Advanced Functional Programming (AFP'09)*, pages 230–266, Heijen, The Netherlands, 2009. Springer.

J. A. Painter. Semantic correctness of a compiler for an Algol-like language. *Stanford Artificial Intelligence Memo. No. 44*, 1(1):1–260, 1967.

Daejun Park, Andrei Ştefănescu, and Grigore Roşu. KJS: A complete formal semantics of JavaScript. In *Proceedings of the 36th annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15)*, pages 346–356, Portland, OR, 2015. ACM.

Andrew Pitts. Construction of the initial algebra for a strictly positiveendofunctor on Set using uniqueness of identity proofs, functionextensionality, quotients types and sized types. Available at `www.cl.cam.ac.uk/users/amp12/agda/initial-T-algebras`., November 2019.

Axel Poigné. Parametrization for order-sorted algebraic specification. *J. Comput. Syst. Sci.*, 40(2):229–268, 1990. doi: 10.1016/0022-0000(90)90013-B. URL `https://doi.org/10.1016/0022-0000(90)90013-B`.

Arthur Prior. *Time and modality.* Greenwood Press, California, USA, 1955.

John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS'02)*, pages 55–74, Copenhagen, Denmark, 2002. IEEE.

Camilo Rocha and José Meseguer. Constructors, sufficient completeness, and deadlock freedom of rewrite theories. In Christian G. Fermüller and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 594–609, Berlin, Heidelberg, 2010. Springer. ISBN 978-3-642-16242-8. doi: 10.1007/978-3-642-16242-8_42.

Grigore Roşu. Matching logic. *Logical Methods in Computer Science*, 13(4):1–61, 2017. doi: 10.23638/LMCS-13(4:28)2017.

Grigore Roşu and Wolfram Schulte. Matching logic—extended report. Technical Report Department of Computer Science UIUCDCS-R-2009-3026, University of Illinois at Urbana-Champaign, January 2009.

Jan J. M. M. Rutten and Daniele Turi. Initial algebra and final coalgebra semantics for concurrency. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium, Noordwijkerhout, The Netherlands, June 1-4, 1993, Proceedings*, volume 803 of *Lecture Notes in Computer Science*, pages 530–582, Noordwijkerhout, The Netherlands, 1993. Springer. doi: 10.1007/3-540-58043-3_28. URL `https://doi.org/10.1007/3-540-58043-3_28`.

Donald Sannella and Andrzej Tarlecki. *Foundations of algebraic specification and formal software development.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, Berlin Heidelberg, Germany, 2012. ISBN 978-3-642-17335-6. doi: 10.1007/978-3-642-17336-3. URL `https://doi.org/10.1007/978-3-642-17336-3`.

Dana Scott. Domains for denotational semantics. In *International Colloquium on Automata, Languages, and Programming*, pages 577–610, Berlin Heidelberg, Germany, 1982. Springer, Springer.

Kristina Sojakova. Higher inductive types as homotopy-initial algebras. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'15)*, POPL '15, pages 31–42, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333009. doi: 10.1145/2676726.2676983. URL `https://doi.org/10.1145/2676726.2676983`.

Bas Spitters and Eelis van der Weegen. Type classes for mathematics in type theory. *Interactive Theorem Proving and the Formalisation of Mathematics*, 21(4):795–825, 2011. doi: 10.1017/S0960129511000119.

Ramesh Subrahmanyam. Complexity of algebraic specifications. In Kesav V. Nori and C. E. Veni Madhavan, editors, *Foundations of Software Technology and Theoretical Computer Science*, pages 33–47, Berlin, Heidelberg, 1990. Springer. doi: 10.1007/3-540-53487-3_33.

Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5 (2):285–309, 1955.

Mark van den Brand, Arie van Deursen, Jan Heering, Hayco de Jong, Merijn de Jonge, Tobias Kuipers, Paul Klint, Leon Moonen, Pieter A. Olivier, Jeroen Scheerder, Jurgen J. Vinju, Eelco Visser, and Joost Visser. The ASF+SDF meta-environment: a component-based language development environment. *Electronic Notes in Theoretical Computer Science*, 44(2):3–8, 2001. doi: 10.1016/S1571-0661(04)80917-4. URL `https://doi.org/10.1016/S1571-0661(04)80917-4`.

H. Wang. Logic of many-sorted theories. *J. Symb. Log.*, 17:105–116, 1952.

A Matching Logic in One Page

The entire metatheory of matching logic is very small and can be put in one page; see Fig. 17.

Signature: a countable set Σ

Variables: two disjoint and countably infinite sets EV and SV

Syntax: $\varphi ::= x \in EV \mid X \in SV \mid \sigma \in \Sigma \mid \varphi_1\ \varphi_2 \mid \bot \mid \varphi_1 \to \varphi_2 \mid \exists x.\, \varphi \mid \mu X.\, \varphi$ (if φ positive in X)

Model: a nonempty carrier set M,
an application interpretation $\_\bullet\_ \colon M \times M \to \mathcal{P}(M)$
a symbol interpretation $\sigma_M \subseteq M$ for each $\sigma \in \Sigma$

Valuation: a function ρ such that $\rho(x) \in M$ for $x \in EV$ and $\rho(X) \subseteq M$ for $X \in SV$

Interpretation: a function $|\_|_\rho$ from patterns to subsets of M

$$|x|_\rho = \{\rho(x)\} \qquad |X|_\rho = \rho(X)$$
$$|\sigma|_\rho = \sigma_M \qquad |\varphi_1\ \varphi_2|_\rho \bigcup_{a \in |\varphi_1|_\rho} \bigcup_{b \in |\varphi_2|_\rho} a \bullet b$$
$$|\bot|_\rho = \emptyset \qquad |\varphi_1 \to \varphi_2|_\rho = M \setminus \left(|\varphi_1|_\rho \setminus |\varphi_2|_\rho\right)$$
$$|\exists x.\, \varphi|_\rho = \bigcup_{a \in M} |\varphi|_{\rho[a/x]} \qquad |\mu X.\, \varphi|_\rho = \bigcap \left\{A \subseteq M \mid |\varphi|_{\rho[A/X]} \subseteq A\right\}$$

Proof System:

FOL Reasoning	(Propositional Tautology)	φ if φ is a tautology over patterns
	(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \to \varphi_2}{\varphi_2}$
	($\exists$-Quantifier)	$\varphi[y/x] \to \exists x.\, \varphi$
	($\exists$-Generalization)	$\dfrac{\varphi_1 \to \varphi_2}{(\exists x. \varphi_1) \to \varphi_2}$ if $x \notin FV(\varphi_2)$
Frame Reasoning	(Propagation$_\bot$)	$C[\bot] \to \bot$
	(Propagation$_\vee$)	$C[\varphi_1 \vee \varphi_2] \to C[\varphi_1] \vee C[\varphi_2]$
	(Propagation$_\exists$)	$C[\exists x.\, \varphi] \to \exists x.\, C[\varphi]$ if $x \notin FV(C)$
	(Framing)	$\dfrac{\varphi_1 \to \varphi_2}{C[\varphi_1] \to C[\varphi_2]}$
Fixpoint Reasoning	(Set Variable Substitution)	$\dfrac{\varphi}{\varphi[\psi/X]}$
	(PreFixpoint)	$\varphi[(\mu X.\, \varphi)/X] \to \mu X.\, \varphi$
	(Knaster-Tarski)	$\dfrac{\varphi[\psi/X] \to \psi}{\mu X.\, \varphi \to \psi}$
Technical Rules	(Existence)	$\exists x.\, x$
	(Singleton)	$\neg\,(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$

where $C[\varphi]$ denotes the application pattern $(\varphi\ \psi)$ or $(\psi\ \varphi)$, for any ψ.

Figure 17: The entire metatheory of matching logic (model theory and proof theory) can be put in one page.

B Proofs of the Results in Section 4

B.1 Proofs of Propositions 4.6, 4.9, and 4.11

Proposition 4.6. *The following propositions hold:*

1. $|\neg\varphi|_{M,\rho} = M \setminus |\varphi|_{M,\rho}$;
2. $|\varphi_1 \vee \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \cup |\varphi_2|_{M,\rho}$;
3. $|\varphi_1 \wedge \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \cap |\varphi_2|_{M,\rho}$;
4. $|\top|_{M,\rho} = M$;
5. $|\varphi_1 \leftrightarrow \varphi_2|_{M,\rho} = M \setminus (|\varphi_1|_{M,\rho} \triangle |\varphi_2|_{M,\rho})$;
6. $|\forall x.\, \varphi|_{M,\rho} = \bigcap_{a\in M} |\varphi|_{\rho[a/x]}$;
7. $|\nu X.\, \varphi|_{M,\rho} = \bigcup\{A \subseteq M \mid A \subseteq |\varphi|_{M,\rho[A/X]}\}$;

where "$\triangle$" denotes set symmetric difference.

Proof. The proofs of (1)-(6) can be found in [Chen and Roşu, 2020b, Appendix A.1]. For (7), we have the following reasoning:

$$\begin{aligned}
|\nu X.\, \varphi|_{M,\rho} &= |\neg\mu X.\, \neg\varphi[\neg X/X]|_{M,\rho} \\
&= M \setminus |\mu X.\, \neg\varphi[\neg X/X]|_{M,\rho} \\
&= M \setminus \bigcap\{A \subseteq M \mid |\neg\varphi[\neg X/X]|_{M,\rho[A/X]} \subseteq A\} \\
&= M \setminus \bigcap\{A \subseteq M \mid (M \setminus A) \subseteq |\varphi[\neg X/X]|_{M,\rho[A/X]}\} \\
&= M \setminus \bigcap\{A \subseteq M \mid (M \setminus A) \subseteq |\varphi|_{M,\rho[(M\setminus A)/X]}\} \\
&= M \setminus \bigcap\{B \subseteq M \mid B \subseteq |\varphi|_{M,\rho[B/X]}\} \\
&= \bigcup\{A \subseteq M \mid A \subseteq |\varphi|_{M,\rho[A/X]}\}.
\end{aligned}$$

□

Proposition 4.9. *For any $M \vDash$ DEFINEDNESS, pattern φ, and valuation ρ:*

1. $\lceil a \rceil_M = M$ *for every $a \in M$, where $\lceil a \rceil_M = def_M \,\bar{\cdot}\, \{a\}$ and def_M is the interpretation of def;*
2. $|\lceil\varphi\rceil|_{M,\rho} = M$ *if $|\varphi|_{M,\rho} \neq \emptyset$; otherwise, $|\lceil\varphi\rceil|_{M,\rho} = \emptyset$.*

Proof. (1). For any element $a \in M$, we consider an M-valuation ρ_a such that $\rho_a(x) = a$ for some variable x. Since $M \vDash \forall x.\, \lceil x \rceil$, we know that $M \vDash \lceil x \rceil$ for all valuations ρ. In particular, we have $|\lceil x \rceil|_{M,\rho_a} = def_M \,\bar{\cdot}\, \{a\} = M$. Therefore, by definition, $\lceil a \rceil_M = M$ for every $a \in M$.

(2). If $|\varphi|_{M,\rho} \neq \emptyset$, there exists $a \in |\varphi|_{M,\rho}$. Therefore, by pointwise extension, $\lceil a \rceil_M \subseteq |\lceil\varphi\rceil|_{M,\rho}$. By (1), $\lceil a \rceil_M = M$, so $|\lceil\varphi\rceil|_{M,\rho} = M$. If $|\varphi|_{M,\rho} = \emptyset$, then by pointwise extension $|\lceil\varphi\rceil|_{M,\rho} = \emptyset$. □

Proposition 4.11. *For any $M \vDash$ NN, the following properties hold:*

1. *By axiom* (Nat Zero)*: $\mathbb{N}_M \neq \emptyset$; $zero_M$ is a singleton; and $zero_M \subseteq \mathbb{N}_M$; thus, we can define $M_{zero} \in \mathbb{N}_M$ to be the unique element that is in $zero_M$;*
2. *By axiom* (Nat Succ)*: for any $m \in \mathbb{N}_M$ there exists $next_m \in \mathbb{N}_M$ such that $succ_M \,\bar{\cdot}\, \{m\} = \{next_m\}$; thus, we can define $M_{succ} \colon \mathbb{N}_M \to \mathbb{N}_M$ to be the unique function such that $M_{succ}(m) = next_m$;*
3. *By axioms* (Nat Succ.1) *and* (Nat Succ.2)*: $M_{succ}(M_{zero}) \neq M_{zero}$, and for any $m, n \in \mathbb{N}_M$, $M_{succ}(m) = M_{succ}(n)$ implies $m = n$, that is, M_{succ} is injective;*

4. *By axiom* (Nat Domain)*: $\mathbb{N}_M$ is the set $\{M_{zero}, M_{succ}(M_{zero}), M_{succ}(M_{succ}(M_{zero})), \dots\}$;*

5. *Thus, $(\mathbb{N}_M, M_{zero}, M_{succ})$ is the standard model of natural numbers.*

Proof. (1). By (Nat Zero), we have $M \vDash \exists x.\, x \in \mathbb{N} \wedge zero = x$. Therefore, there exists $a \in \mathbb{N}_M$ such that $zero_M = \{a\}$. Thus, $\mathbb{N}_M \neq \emptyset$, and $zero_M$ is a singleton.

(2). By (Nat Succ), we have $M \vDash \forall x.\, x \in \mathbb{N} \to \exists y.\, y \in \mathbb{N} \wedge succ\ x = y$. Therefore, for all $m \in \mathbb{N}_M$ there exists $next_m \in \mathbb{N}_M$, such that $succ_M \bar{\bullet} \{m\} = \{next_m\}$, and clearly such $next_m$ is unique. Therefore, the function $M_{succ} \colon \mathbb{N}_M \to \mathbb{N}_M$ defined by $M_{succ}(m) = next_m$ for all $m \in \mathbb{N}_M$ is indeed well-defined.

(3). The conclusion holds trivially.

(4). We verify that the set $\{M_{zero}, M_{succ}(M_{zero}), M_{succ}(M_{succ}(M_{zero})), \dots\}$, denoted as $\mathbb{N}'_M$ for now, is indeed the least fixpoint of function $\mathcal{F} \colon \mathcal{P}(M) \to \mathcal{P}(M)$ defined by $\mathcal{F}(A) = zero_M \cup (succ_M \bar{\bullet} A)$ for all $A \subseteq M$. Firstly, we verify that $\mathbb{N}'_M$ is a fixpoint of $\mathcal{F}$ as follows:

$$\begin{aligned}
\mathcal{F}(\mathbb{N}'_M) &= zero_M \cup (succ_M \bar{\bullet} \mathbb{N}'_M) \\
&= \{M_{zero}\} \cup \bigcup_{a \in \mathbb{N}'_M} succ_M \bar{\bullet} \{a\} \\
&= \{M_{zero}\} \cup \{M_{succ}(M_{zero}), M_{succ}(M_{succ}(M_{zero})), \dots\} \\
&= \mathbb{N}'_M.
\end{aligned}$$

Secondly, we show that any strict subset $\mathbb{N}^0_M \subsetneq \mathbb{N}'_M$ is not a fixpoint of $\mathcal{F}$, and thus $\mathbb{N}'_M$ is indeed the least fixpoint of $\mathcal{F}$. Let $k \geq 0$ be the smallest k such that $M^k_{succ}(M_{zero}) \notin \mathbb{N}^0_M$, where $M^k_{succ}(M_{zero}) = \underbrace{M_{succ}(M_{succ}(\dots M_{succ}}_{k \text{ times}}(M_{zero})\dots))$. If $k = 0$, then $M_{zero} \notin \mathbb{N}^0_M$, and thus $\mathbb{N}^0_M$ is not a fixpoint of $\mathcal{F}$. If $k > 0$, then we have $M^{k-1}_{succ}(M_{zero}) \in \mathbb{N}^0_M$ but $M^k_{succ}(M_{zero}) \notin \mathbb{N}^0_M$, which also shows that $\mathbb{N}^0_M$ is not a fixpoint of $\mathcal{F}$. Therefore, $\mathbb{N}'_M$ is indeed the least fixpoint of $\mathcal{F}$, and thus we have $\mathbb{N}_M = \mathbb{N}'_M = \{M_{zero}, M_{succ}(M_{zero}), M_{succ}(M_{succ}(M_{zero})), \dots\}$. □

B.2 Matching Logic Proof System

The proof system of matching logic is shown in Fig. 17, where we draw inspiration from [Chen and Roşu, 2019a, 2020a]. It consists of four modules: FOL reasoning, equational reasoning, fixpoint reasoning, and some technical rules. Particularly, the fixpoint reasoning module consists of three rules. (Set Variable Substitution) allows one to substitute any pattern ψ for a set variable X in a (formal) theorem. (PreFixpoint) states that $\mu X.\, \varphi$ is a *pre-fixpoint*, in the sense that the unfolded pattern $\varphi[(\mu X.\, \varphi)/X]$ is included by original fixpoint $\mu X.\, \varphi$. (Knaster-Tarski) is the logical incarnation of the Knaster-Tarski fixpoint theorem.

B.3 Proofs of Proposition 4.12 and Theorem 4.13

Proposition 4.12. *Let Γ be any specification. Then, the following propositions hold:*

1. *$\Gamma \vdash \varphi$, if φ is a tautology over patterns;*

2. *$\Gamma \vdash \varphi_1$ and $\Gamma \vdash \varphi_1 \to \varphi_2$ imply $\Gamma \vdash \varphi_2$;*

3. *$\Gamma \vdash \varphi[y/x] \to \exists x.\, \varphi$;*

4. *$\Gamma \vdash \varphi_1 \to \varphi_2$ and $y \notin FV(\varphi_2)$ imply $\Gamma \vdash (\exists y.\, \varphi_1) \to \varphi_2$;*

5. *$\Gamma \vdash \varphi = \varphi$;*

6. *$\Gamma \vdash \varphi_1 = \varphi_2$ and $\Gamma \vdash \varphi_2 = \varphi_3$ imply $\Gamma \vdash \varphi_1 = \varphi_3$;*

7. *$\Gamma \vdash \varphi_1 = \varphi_2$ implies $\Gamma \vdash \varphi_2 = \varphi_1$;*

8. $\Gamma \vdash \varphi_1 = \varphi_2$ *implies* $\Gamma \vdash \psi[\varphi_1/x] = \psi[\varphi_2/x]$*, known as the Leibniz's law of equality.*

9. $\Gamma \vdash (\mu X.\, \varphi) = \varphi[\mu X.\, \varphi/X]$*;*

10. $\Gamma \vdash \varphi[\psi/X] \to \psi$ *implies* $\Gamma \vdash (\mu X.\, \varphi) \to \psi$*; this proof rule is denoted* (Knaster-Tarski)*;*

where for (5)-(9) we naturally require that Γ *defines equality (Section 4.3.1).*

Proof. The proofs of (1)-(8) can be found in [Chen and Roşu, 2020b, Appendix B.3]. (10) is exactly the proof rule (Knaster-Tarski) in Fig. 17. Therefore, we only need to prove (9), which is equivalent to $\Gamma \vdash \lfloor (\mu X.\, \varphi) \leftrightarrow \varphi[\mu X.\, \varphi/X] \rfloor$. The implication from right to left is the proof rule (PreFixpoint). Therefore, we only need to prove the other direction: $\Gamma \vdash (\mu X.\, \varphi) \to \varphi[(\mu X.\, \varphi)/X]$.

We apply rule (Knaster-Tarski) and obtain the proof goal $\Gamma \vdash \varphi[(\varphi[(\mu X.\, \varphi)/X])/X] \to \varphi[(\mu X.\, \varphi)/X]$. Note that φ is positive in X, and thus by frame reasoning (see, e.g., [Chen and Roşu, 2019b, Lemma 89]), we only need to prove that $\varphi[(\mu X.\, \varphi)/X] \to \mu X.\, \varphi$, which is proved by (Pre-Fixpoint). ☐

Theorem 4.13. *For any* Γ *and* φ*, we have* $\Gamma \vdash \varphi$ *implies* $\Gamma \vDash \varphi$*.*

Proof. We only need to show that all 13 proof rules listed in Fig. 17 are sound. Here we only show the proofs for the fixpoint reasoning rules, (PreFixpoint) and (Knaster-Tarski), as the rest can be found in [Chen and Roşu, 2020b, Appendix B.3].

For a model M and a valuation ρ, the pattern φ defines (w.r.t. X) a monotone function $\mathcal{F} \colon \mathcal{P}(M) \to \mathcal{P}(M)$ given by $\mathcal{F}(A) = |\varphi|_{\rho[A/X]}$. By the Knaster-Tarski fixpoint theorem, the least fixpoint of $\mathcal{F}$, denoted $\mu\mathcal{F}$, is given as:

$$\mu\mathcal{F} = \bigcap \{A \subseteq M \mid \mathcal{F}(A) \subseteq A\}.$$

For (PreFixpoint), we have that $|\varphi[(\mu X.\, \varphi)/X]|_{M,\rho} = |\varphi|_{M,\rho[(|\mu X.\, \varphi|_{M,\rho})/X]} = \mathcal{F}(|\mu X.\, \varphi|_{M,\rho}) = \mathcal{F}(\mu\mathcal{F})$, which equals to $\mu\mathcal{F}$ because it is a fixpoint of $\mathcal{F}$. For (Knaster-Tarski), let us consider any pattern ψ such that $\vdash \varphi[\psi/X] \to \psi$. Let $B = |\psi|_{M,\rho}$, then we have $|\varphi[\psi/X]|_{M,\rho} \subseteq B$, which implies that $\mathcal{F}(B) \subseteq B$. Therefore, (Knaster-Tarski) is sound. ☐

C Proofs of the Results in Section 5

Proposition 5.2. *The following propositions hold:*

1. *By* (Nat Sort)*:* Nat_M *is a singleton, whose unique element we (ambiguously) denote also as* Nat_M*; then,* $Nat_M \in M_{Sort}$*; intuitively,* Nat_M *is the interpretation of the sort name Nat in* M*;*

2. *By* (Nat)*: the set* M_{Nat} *equals* $\mathbb{N}_M = \{M_{zero}, M_{succ}(M_{zero}), M_{succ}(M_{succ}(M_{zero})), \dots\}$*, which was defined in Proposition 4.11; intuitively,* M_{Nat} *is the inhabitant set of Nat in* M*;*

3. *Following the same reasoning in Proposition 4.11, we can define functions* $M_{plus}, M_{mult} \colon M_{Nat} \times M_{Nat} \to M_{Nat}$*, such that* $plus_M \mathbin{\bar{\bullet}} \{m\} \mathbin{\bar{\bullet}} \{n\} = \{M_{plus}(m,n)\}$ *and* $mult_M \mathbin{\bar{\bullet}} \{m\} \mathbin{\bar{\bullet}} \{n\} = \{M_{mult}(m,n)\}$*, for all* $m, n \in M_{Nat}$*; that is, they capture the addition and multiplication functions.*

Proof. The proofs of (1) and (3) follow the same idea as the proof of Proposition 4.11, and (2) follows directly from Proposition 4.11. ☐

Proposition 5.3. *Under the above conditions and notations, the following properties hold:*

1. *By* (Pair Sort)*: for* $s_1, s_2 \in M_{Sort}$*,* $Pair_M \mathbin{\bar{\bullet}} \{s_1\} \mathbin{\bar{\bullet}} \{s_2\}$ *is a singleton, whose (unique) element we denote* $s_1 \otimes_M s_2$*; then we have* $s_1 \otimes_M s_2 \in M_{Sort}$*; intuitively,* $s_1 \otimes_M s_2$ *is the pair sort of* s_1 *and* s_2*;*

2. *By* (Pair)*: for* $s_1, s_2 \in M_{Sort}$*,* $x_1 \in M_{s_1}$*, and* $x_2 \in M_{s_2}$*,* $pair_M \mathbin{\bar{\bullet}} \{x_1\} \mathbin{\bar{\bullet}} \{x_2\}$ *is a singleton, whose (unique) element we denote* $\langle x_1, x_2 \rangle_M$*; then we have* $\langle x_1, x_2 \rangle_M \in M_{s_1 \otimes_M s_2}$*;*

3. *Like in Propositions 4.3.2 and 5.1.1, let $M_{fst}\colon M_{s_1 \otimes_M s_2} \to M_{s_1}$ and $M_{snd}\colon M_{s_1 \otimes_M s_2} \to M_{s_2}$ be such that $fst_M \mathbin{\bar{\bullet}} \{y\} = \{M_{fst}(y)\}$ and $snd_M \mathbin{\bar{\bullet}} \{y\} = \{M_{snd}(y)\}$, for any $s_1, s_2 \in M_{Sort}$ and $y \in M_{s_1 \otimes_M s_2}$;*
4. *By* (Pair Fst/Snd/Inj)*: for $s_1, s_2 \in M_{Sort}$, $x_1, y_1 \in M_{s_1}$, and $x_2, y_2 \in M_{s_2}$, we have $M_{fst}(\langle x_1, x_2\rangle_M) = x_1$, $M_{snd}(\langle x_1, x_2\rangle_M) = x_2$, and $\langle x_1, x_2\rangle_M = \langle y_1, y_2\rangle_M$ implies $x_i = y_i$, $(i = 1, 2)$;*
5. *$M_{s_1 \otimes_M s_2} = \{\langle x_1, x_2\rangle_M \mid x_1 \in M_{s_1}, x_2 \in M_{s_2}\}$;*
6. *Then, $M_{s_1 \otimes_M s_2}$ is exactly the Cartesian product $M_{s_1} \times M_{s_2}$ of M_{s_1} and M_{s_2}, for $s_1, s_2 \in M_{Sort}$.*

Proof. The proofs of (1)-(3) are the same as the proof of Proposition 4.11 and (4) follows directly from the axioms and Proposition 4.6. (5) follows directly from the axiom (Pair Domain), and (6) follows from the bijection function $i\colon M_{s_1} \times M_{s_2} \to M_{s_1 \otimes s_2}$, defined by $i(x_1, x_2) = \langle x_1, x_2\rangle_M$ for all $x_1 \in M_{s_1}$ and $x_2 \in M_{s_2}$. The injectivity of i is guaranteed by axiom (Pair Injectivity) and the surjectivity of i is guaranteed by axiom (Pair Domain). $\square$

Proposition 5.5. *The following holds for any $n \geq 0$:*

$$\mathsf{PRELUDE} \vdash \forall s_1, \ldots, s_n {:} Sort.\ (f\colon s_1 \otimes \cdots \otimes s_n \ominus s) \to (\forall x_1 {:} s_1. \ldots \forall x_n {:} s_n. f(x_1, \ldots, x_n) {:} s)$$

Proof. Recall that $f{:}s_1 \otimes \cdots \otimes s_n \ominus s$ is sugar for $\exists g{:}s_1 \otimes \cdots \otimes s_n \ominus s \wedge f = g$ (see Definition 5), and the latter further desugars to $\exists g.\, (g \in s_1 \otimes \cdots \otimes s_n \ominus s) \wedge f = g$. By FOL reasoning, we have $f \in s_1 \otimes \cdots \otimes s_n \ominus s$, and by axiom (Function Domain), we have the intended property $\forall x_1 {:} s_1. \ldots \forall x_n {:} s_n. f(x_1, \ldots, x_n) {:} s$. $\square$

Proposition 5.7. *Let* SPEC *be a specification that imports* $\mathsf{PRELUDE}$ *and includes the above axiom* (Function) *for a symbol f. Then f yields a function $M_f\colon M_{s_1} \times \cdots \times M_{s_n} \to M_s$ in any $M \vDash \mathsf{SPEC}$.*

Proof. The proof follows the same idea as the proof of Proposition 4.11. Due to the importance of this proposition, we re-state the proof as follows.

Let f_M be the interpretation of f in M. For each sort $t \in \{s_1, \ldots, s_n, s\}$, let t_M be its interpretation and $M_t = inh_M \mathbin{\bar{\bullet}} t_M$ be the inhabitant set of t in M. Then, for any $x_i \in M_{s_i}$, $1 \leq i \leq n$, the set $f_M \mathbin{\bar{\bullet}} \{\langle x_1, \ldots, x_n\rangle_M\}$ is a singleton, whose (unique) element we denote $M_f(x_1, \ldots, x_n)$. Thus, f yields a function $M_f\colon M_{s_1} \times \cdots \times M_{s_n} \to M_s$. $\square$

Note that axiom (Function) also enforces the interpretation f_M of the symbol f in M to be a singleton, containing exactly the "function object". This fact is not needed to prove the subsequent results, so we do not state it in the above proposition, but it helps to build the intuition for the matching logic specification of functions.

D Proofs of the Results in Section 6

Here, we study the matching logic specifications $\mathsf{ALGEBRA}(S, F)$ of F-algebras, and also the matching logic specifications $\mathsf{EQSPEC}(S, F, E)$ of E-algebras. In Appendix D.1, we show the construction of the standard matching logic models of $\mathsf{ALGEBRA}(S, F)$. In Appendix D.2, we study the semantic equivalence between the matching logic models and the derived algebras. In Appendix D.3, we capture the equivalence relation using institution comorphisms, a categorical notion for logic embeddings developed in [Goguen and Rosu, 2002].

D.1 Standard Models of $\mathsf{ALGEBRA}(S, F)$

Throughout this section, we assume and fix an (arbitrary) many-sorted signature (S, F) and an (S, F)-algebra A. We will construct a corresponding matching logic model M, which we call the standard model (w.r.t. A). Intuitively, M interprets the carrier sets of sorts $s \in S$ the same as A, and interprets operations the same as the operation interpretations of A. Our goal is to prove that the model transformation α is essentially surjective (Theorem 6.5). As said before, the most technically tedious work is to construct the nested pair and function sorts defined in Section 5 when we construct the standard model M.

D.1.1 A Summary of $\mathsf{ALGEBRA}(S, F)$

For representational purpose, we show the entire specification $\mathsf{ALGEBRA}(S, F)$ in one place in Fig. 18, where all imported specifications are expanded inline. We omit notation definitions for simplicity.

spec $\mathsf{ALGEBRA}(S, F)$

Symbols: $\mathit{def}, \mathit{inh}, \mathit{Sort}, \mathbb{N}, \mathit{Nat}, \mathit{zero}, \mathit{succ}, \mathit{plus}, \mathit{mult}, \mathit{NzNat}, \mathit{Pair}, \mathit{pair}, \mathit{fst}, \mathit{snd}, \mathit{proj}, \mathit{Function}$ $\mathit{Unit}, \mathit{unit}, s \in S, f \in F, \mathit{SigOps}, \mathit{SigArgs}$

Notations: omitted

Axioms:

(Definedness)	$\forall x.\, \lceil x \rceil$
(Nat Zero)	$\exists x.\, x \in \mathbb{N} \wedge \mathit{zero} = x$
(Nat Succ)	$\forall x.\, x \in \mathbb{N} \to \exists y.\, y \in \mathbb{N} \wedge \mathit{succ}\ x = y$
(Nat Succ.1)	$\mathit{succ}\ \mathit{zero} \neq \mathit{zero}$
(Nat Succ.2)	$\forall x.\, \forall y.\, x \in \mathbb{N} \wedge y \in \mathbb{N} \to \mathit{succ}\ x = \mathit{succ}\ y \to x = y$
(Nat Domain)	$\mathbb{N} = \mu D.\, \mathit{zero} \vee \mathit{succ}\ D$
(Nat Sort)	$\mathit{Nat} : \mathit{Sort}$
(Nat)	$[\![\mathit{Nat}]\!] = \mathbb{N}$
(Nat Plus.1)	$\forall x : \mathit{Nat}.\, \mathit{plus}\ x\ \mathit{zero} = x$
(Nat Plus.2)	$\forall x, y : \mathit{Nat}.\, \mathit{plus}\ x\ (\mathit{succ}\ y) = \mathit{succ}\ (\mathit{plus}\ x\ y)$
(Nat Mult.1)	$\forall x : \mathit{Nat}.\, \mathit{mult}\ x\ \mathit{zero} = \mathit{zero}$
(Nat Mult.2)	$\forall x, y : \mathit{Nat}.\, \mathit{mult}\ x\ (\mathit{succ}\ y) = \mathit{plus}\ x\ (\mathit{mult}\ x\ y)$
(NzNat Sort)	$\mathit{NzNat} : \mathit{Sort}$
(NzNat)	$[\![\mathit{NzNat}]\!] = \mathit{succ}\ [\![\mathit{Nat}]\!]$
(Pair Sort)	$\forall s_1, s_2 : \mathit{Sort}.\, (s_1 \otimes s_2) : \mathit{Sort}$
(Pair)	$\forall s_1, s_2 : \mathit{Sort}.\, \forall x_1 : s_1.\, \forall x_2 : s_2.\ \langle x_1, x_2 \rangle : (s_1 \otimes s_2)$
(Pair Fst)	$\forall s_1, s_2 : \mathit{Sort}.\, \forall x_1 : s_1.\, \forall x_2 : s_2.\, \mathit{fst}\ \langle x_1, x_2 \rangle = x_1$
(Pair Snd)	$\forall s_1, s_2 : \mathit{Sort}.\, \forall x_1 : s_1.\, \forall x_2 : s_2.\, \mathit{snd}\ \langle x_1, x_2 \rangle = x_2$
(Pair Inj)	$\forall s_1, s_2 : \mathit{Sort}.\, \forall x_1, y_1 : s_1.\, \forall x_2, y_2 : s_2.\ \langle x_1, x_2 \rangle = \langle y_1, y_2 \rangle \to x_1 = x_2 \wedge y_1 = y_2$
(Pair Domain)	$\forall s_1, s_2 : \mathit{Sort}.\, [\![s_1 \otimes s_2]\!] = \langle [\![s_1]\!], [\![s_2]\!] \rangle$
(Proj First)	$\forall s_1, s_2 : \mathit{Sort}.\, \forall x_1 : s_1.\, \forall x_2 : s_2.\, (\mathit{proj}\ 1\ \langle x_1, x_2 \rangle) = x_1$
(Proj Rest)	$\forall s_1, s_2 : \mathit{Sort}.\, \forall x_1 : s_1.\, \forall x_2 : s_2.\, \forall n : \mathit{NzNat}.\, (\mathit{proj}\ (\mathit{succ}\ n)\ \langle x_1, x_2 \rangle) = \mathit{proj}\ n\ x_2$
(Function Sort)	$\forall s_1, s_2 : \mathit{Sort}.\, (s_1 \ominus s_2) : \mathit{Sort}$
(Function Domain)	$\forall s_1, s_2 : \mathit{Sort}.\, [\![s_1 \ominus s_2]\!] = \exists f.\, f \wedge (\forall x : s_1.\, (f\ x) : s_2)$
(Function Ext)	$\forall s_1, s_2 : \mathit{Sort}.\, \forall f, g : s_1 \ominus s_2.\, (\forall x : s_1.\, f\ x = g\ x) \to f = g$
(Unit Sort)	$\mathit{Unit} : \mathit{Sort}$
(Unit)	$\mathit{unit} : \mathit{Unit}$
(Unit Domain)	$[\![\mathit{Unit}]\!] = \mathit{unit}$
(Unit Identity)	$\forall s : \mathit{Sort}.\, \forall x : s.\, (x\ \mathit{unit}) = x$
(Sort)	$s : \mathit{Sort}$ for each $s \in S$
(Function)	$f : s_1 \otimes \cdots \otimes s_n \ominus s$ for each $f \in F_{s_1 \ldots s_n, s}$
(Signature Operation)	$[\![\mathit{SigOps}]\!] = \bigvee_{f \in F} f$
(Signature ArgTuple)	$[\![\mathit{SigArgs}]\!] = \bigvee_{f \in F_{s_1 \ldots s_n, s}} [\![s_1 \otimes \cdots \otimes s_n]\!]$

endspec

Figure 18: The entire matching logic specification $\mathsf{ALGEBRA}(S, F)$ is shown in one place; the specification $\mathsf{INITIALALGEBRA}(S, F, E)$ of initial algebras is extended by axiom (No Junk); axiom (No Confusion); and axioms that define the E-equivalence classes like in Section 8, which can also fit in one page.

D.1.2 Constructing the Carrier Set of M

At a high level, the carrier set M includes both the regular elements and elements that represent sort names, functions, operations, and predicates. We first introduce some definitions and notations.

Definition D.1. For a finite set $\mathbb{S}$ of sets, we define $\mathbb{S}^*$ as the smallest set such that:

1. $\mathbb{S} \subseteq \mathbb{S}^*$;

2. $S_1 \times S_2 \in \mathbb{S}^*$ if $S_1, S_2 \in \mathbb{S}^*$.

Therefore, $\mathbb{S}^*$ is the smallest set that includes $\mathbb{S}$ and is closed under the Cartesian product/square.

Definition D.2. For a finite set $\mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}$ whose elements we call *sort names*, we define $\mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}^*$ as the smallest set that satisfies the following conditions:

1. $\mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame} \subseteq \mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}^*$;
2. $s \otimes s' \in \mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}^*$, if $s, s' \in \mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}$;
3. $s \ominus s' \in \mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}^*$, if $s, s' \in \mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}$.

Therefore, $\mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}^*$ is the smallest set that includes $\mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}$ and is closed under the construction of pair and function sorts.

Recall that we have assumed and fixed an (S, F)-algebra A at the beginning of the section. The carrier sets of A are denoted A_s for each sort $s \in S$, and operation interpretations are denoted $A_f\colon M_{s_1} \times \cdots \times M_{s_n} \to M_s$ for each $f \in F_{s_1 \ldots s_n, s}$.

Definition D.3. Let M be the disjoint union of the following sets:

1. $\{\#\mathsf{def}, \#\mathsf{inh}, \#\mathsf{Sort}, \#\mathsf{Nat}, \#\mathsf{zsucc}, \#\mathsf{plus}, \#\mathsf{mult}, \#\mathsf{NzNat}, \#\mathsf{Pair}, \#\mathsf{pair}, \#\mathsf{fst}, \#\mathsf{snd}, \#\mathsf{proj}, \#\mathsf{Function}, \#\mathsf{Unit}, \#\mathsf{unit}, \#\mathsf{SigOps}, \#\mathsf{SigArgs}\}$, which include distinguished elements used to interpret the corresponding symbols in $\mathsf{ALGEBRA}(S, F)$;
2. $\{\#\mathsf{s} \mid s \in S\}$, where $\#\mathsf{s}$ is used to interpret the sort name s;
3. $\mathbb{N}$, which is the set of natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$;
4. A_s, for each $s \in S$;
5. each set in $\mathbb{S}^*$, where $\mathbb{S} = \{\mathbb{N}, \mathbb{N}_{>0}\} \cup \{A_s \mid s \in S\}$;
6. $\mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}^*$, where $\mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame} = \{\#\mathsf{Nat}, \#\mathsf{NzNat}, \#\mathsf{Unit}\} \cup \{\#\mathsf{s} \mid s \in S\}$;
7. the set of functions $[A_{s_1} \times \cdots \times A_{s_n} \to A_s]$ if $F_{s_1 \ldots s_n, s} \neq \emptyset$;
8. $\{\#\mathsf{plus} \rightsquigarrow n \mid n \in \mathbb{N}\}$, where $\#\mathsf{plus} \rightsquigarrow n$ is the partial evaluation result of $\#\mathsf{plus}$ on n;
9. $\{\#\mathsf{mult} \rightsquigarrow n \mid n \in \mathbb{N}\}$, where $\#\mathsf{mult} \rightsquigarrow n$ is the partial evaluation result of $\#\mathsf{mult}$ on n;
10. $\{\#\mathsf{pair} \rightsquigarrow a \mid a \in \mathbb{S}^*\}$, where $\#\mathsf{pair} \rightsquigarrow a$ is the partial evaluation result of $\#\mathsf{pair}$ on a;
11. $\{\#\mathsf{Pair} \rightsquigarrow s \mid s \in \mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}^*\}$, where $\#\mathsf{Pair} \rightsquigarrow s$ is the partial evaluation result of $\#\mathsf{Pair}$ on s;
12. $\{\#\mathsf{proj} \rightsquigarrow n \mid n \in \mathbb{N}_{>0}\}$, where $\#\mathsf{proj} \rightsquigarrow n$ is the partial evaluation result of $\#\mathsf{proj}$ on $n > 0$.
13. $\{\#\mathsf{Function} \rightsquigarrow s \mid s \in \mathbb{S}\mathsf{ort}\mathbb{N}\mathsf{ame}^*\}$, where $\#\mathsf{Function} \rightsquigarrow s$ is the partial evaluation result of $\#\mathsf{Function}$ on s.

Here we finish the construction of the carrier set M.

D.1.3 Defining Symbol Interpretation in M

Recall that we use $\sigma_M \subseteq M$ to denote the interpretation of a symbol σ that is defined in $\mathsf{ALGEBRA}(S, F)$.

Definition D.4. We define symbol interpretation in M as follows:

1. $\mathit{def}_M = \{\#\mathsf{def}\}$;
2. $\mathit{inh}_M = \{\#\mathsf{inh}\}$;

3. $Sort_M = \{\#\mathsf{Sort}\}$;
4. $\mathbb{N}_M = \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers;
5. $Nat_M = \{\#\mathsf{Nat}\}$;
6. $zero_M = \{0\}$;
7. $succ_M = \{\#\mathsf{zsucc}\}$;
8. $plus_M = \{\#\mathsf{plus}\}$;
9. $mult_M = \{\#\mathsf{mult}\}$;
10. $NzNat_M = \{\#\mathsf{NzNat}\}$;
11. $Pair_M = \{\#\mathsf{Pair}\}$;
12. $pair_M = \{\#\mathsf{pair}\}$;
13. $fst_M = \{\#\mathsf{fst}\}$;
14. $snd_M = \{\#\mathsf{snd}\}$;
15. $proj_M = \{\#\mathsf{proj}\}$;
16. $Function_M = \{\#\mathsf{Function}\}$;
17. $Unit_M = \{\#\mathsf{Unit}\}$;
18. $unit_M = \{\#\mathsf{unit}\}$;
19. $s_M = \{\#\mathsf{s}\}$, for each $s \in S$;
20. $f_M = \{f_A\}$, for each $f \in F_{s_1 \ldots s_n, s}$, where $f_A \colon A_{s_1} \times \cdots \times A_{s_n} \to A_s$ is the operation interpretation of f in the algebra A; note that if $F_{s_1 \ldots s_n, s} \neq \emptyset$, the function space $[A_{s_1} \times \cdots \times A_{s_n} \to A_s]$ is included in M according to Definition D.3(7);
21. $SigOps_M = \{\#\mathsf{SigOps}\}$;
22. $SigArgs_M = \{\#\mathsf{SigArgs}\}$.

D.1.4 Defining Application Interpretation in M

The application interpretation $\_\bullet\_ \colon M \times M \to \mathcal{P}(M)$ consists of three main parts: (1) the result of applying $\#\mathsf{def}$ to all elements; (2) the result of applying $\#\mathsf{inh}$ to sorts; (3) the (partial) evaluation results of applying functions to their arguments.

Definition D.5. Let $\mathbb{S}^*$ and $\mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ be defined like in Definition D.3(5,6). We define the application interpretation function $\_\bullet\_ \colon M \times M \to M$ as follows:

1. $\#\mathsf{def} \bullet a = M$, for all $a \in M$;
2. $\#\mathsf{inh} \bullet \#\mathsf{Sort} = \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$, where $\mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ is defined in Definition D.3(6);
3. $\#\mathsf{inh} \bullet \#\mathsf{Nat} = \mathbb{N}$;
4. $\#\mathsf{inh} \bullet \#\mathsf{NzNat} = \mathbb{N}_{>0}$;
5. $\#\mathsf{inh} \bullet \#\mathsf{Unit} = \{\#\mathsf{unit}\}$;

6. $\#\mathsf{inh} \bullet \#\mathsf{s} = A_s$;
7. $\#\mathsf{zsucc} \bullet n = \{n+1\}$, for each $n \in \mathbb{N}$;
8. $\#\mathsf{plus} \bullet n = \{\#\mathsf{plus} \rightsquigarrow n\}$, for each $n \in \mathbb{N}$;
9. $(\#\mathsf{plus} \rightsquigarrow n) \bullet m = \{n+m\}$, for each $m, n \in \mathbb{N}$;
10. $\#\mathsf{mult} \bullet n = \{\#\mathsf{mult} \rightsquigarrow n\}$, for each $n \in \mathbb{N}$;
11. $(\#\mathsf{mult} \rightsquigarrow n) \bullet m = \{mn\}$, for each $m, n \in \mathbb{N}$;
12. $\#\mathsf{Pair} \bullet s = \{\#\mathsf{Pair} \rightsquigarrow s\}$, for each $s \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$;
13. $(\#\mathsf{Pair} \rightsquigarrow s) \bullet s' = \{s \otimes s'\}$, for each $s, s' \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$;
14. $\#\mathsf{pair} \bullet a = \{\#\mathsf{pair} \rightsquigarrow a\}$, for each $a \in \mathbb{S}^*$;
15. $(\#\mathsf{pair} \rightsquigarrow a) \bullet b = \{(a, b)\}$, for each $a, b \in \mathbb{S}^*$;
16. $\#\mathsf{fst} \bullet (a, b) = \{a\}$, for each $a, b \in \mathbb{S}^*$;
17. $\#\mathsf{snd} \bullet (a, b) = \{b\}$, for each $a, b \in \mathbb{S}^*$;
18. $\#\mathsf{proj} \bullet n = \{\#\mathsf{proj} \rightsquigarrow n\}$, for each $n \in \mathbb{N}_{>0}$;
19. $(\#\mathsf{proj} \rightsquigarrow i) \bullet (a_1, \ldots, a_i, \ldots, a_n) = a_i$, for $i > 0$ and $a_1, \ldots, a_n \in \mathbb{S}^*$;
20. $\#\mathsf{inh} \bullet (s \otimes s') = (\#\mathsf{inh} \bullet s) \times (\#\mathsf{inh} \bullet s')$, for each $s, s' \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$;
21. $\#\mathsf{Function} \bullet s = \{\#\mathsf{Function} \rightsquigarrow s\}$, for each $s \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$;
22. $(\#\mathsf{Function} \rightsquigarrow s) \bullet s' = \{s \ominus s'\}$, for each $s, s' \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$;
23. $\#\mathsf{inh} \bullet (s_1 \otimes \cdots \otimes s_n \ominus s') = [A_{s_1} \times \cdots \times A_{s_n} \to A_{s'}]$, if $F_{s_1 \ldots s_n, s} \neq \emptyset$; see Definition D.3(7);
24. $\#\mathsf{inh} \bullet \#\mathsf{Unit} = \{\#\mathsf{unit}\}$;
25. $a \bullet \#\mathsf{unit} = a$, for each $a \in \mathbb{S}^*$;
26. $f \bullet (a_1, \ldots, a_n) = f(a_1, \ldots, a_n)$, if $F_{s_1 \ldots s_n, s} \neq \emptyset$, $f \colon A_{s_1} \times \cdots \times A_{s_n} \to A_s$, and $a_i \in A_{s_i}$, $1 \leq i \leq n$; in this case, the function space $[A_{s_1} \times \cdots \times A_{s_n} \to A_s]$ is included in M according to Definition D.3(7);
27. if none of the above applies, we define $a \bullet b = \emptyset$ for $a, b \in M$.

The above definition, although long, is a straightforward description of the behaviors of the definedness symbol *def*, the inhabitant symbol *inh*, sort constructors, and (many-sorted) functions, and therefore has (almost) nothing smart. The only smart thing is the optimization in rule (23), where we only define the inhabitant sets for the function sorts that are indeed in the signature F. For the other (infinitely many) function sorts, we define their inhabitant sets to be empty. This is possible because unlike the pair sort, the function sort has no constructors, so its inhabitants can be freely determined by the model M.

D.1.5 Verifying the Axioms

There are 32 axioms in specification $\mathsf{ALGEBRA}(S, F)$; see Fig. 18. We verify that the model M previously constructed indeed validate these axioms. The verification, listed below, is straightforward.

(Definedness). We have $\{\#\mathsf{def}\} \bullet \{a\} = M$ for any $a \in M$.

(Nat Zero). Recall that $\mathbb{N}_M = \mathbb{N}$ is the set of natural numbers. Thus, we have $0 \in \mathbb{N}$ and by definition, $|\mathbb{0}|_M = \{0\}$.

(Nat Succ). For any $n \in \mathbb{N}$, there exists $m = n + 1$ such that $\#\mathsf{zsucc} \bullet n = \{n + 1\} = \{m\}$.

(Nat Succ.1). By definition, $\#\mathsf{zsucc} \bullet 0 = \{1\} \neq \{0\}$.

(Nat Succ.2). For any $m, n \in \mathbb{N}$ such that $\#\mathsf{zsucc} \bullet m = \#\mathsf{zsucc} \bullet n$, we have $\{m + 1\} = \{n + 1\}$, and thus $m = n$.

(Nat Domain). This has been proved in Proposition 4.11.

(Nat Sort). We have $\#\mathsf{Nat} \in \#\mathsf{inh} \bullet \#\mathsf{Sort}$ by definition.

(Nat). We have $\#\mathsf{inh} \bullet \#\mathsf{Nat} = \mathbb{N}$ by definition.

(Nat Plus.1). For any $n \in \mathbb{N}$, we have $\{\#\mathsf{plus}\} \mathbin{\bar{\bullet}} \{n\} \mathbin{\bar{\bullet}} \{0\} = \{n + 0\} = \{n\}$.

(Nat Plus.2). For any $m, n \in \mathbb{N}$, we have $\{\#\mathsf{plus}\} \mathbin{\bar{\bullet}} \{n\} \mathbin{\bar{\bullet}} (\{\#\mathsf{zsucc}\} \mathbin{\bar{\bullet}} \{m\}) = \{n + m + 1\} = \{\#\mathsf{zsucc}\} \mathbin{\bar{\bullet}} (\{\#\mathsf{plus}\} \mathbin{\bar{\bullet}} \{n\} \mathbin{\bar{\bullet}} \{m\})$.

(Nat Mult.1). For any $n \in \mathbb{N}$, we have $\{\#\mathsf{mult}\} \mathbin{\bar{\bullet}} \{n\} \mathbin{\bar{\bullet}} \{0\} = \{0\}$.

(Nat Mult.2). For any $m, n \in \mathbb{N}$, we have $\{\#\mathsf{mult}\} \mathbin{\bar{\bullet}} \{n\} \mathbin{\bar{\bullet}} (\{\#\mathsf{zsucc}\} \mathbin{\bar{\bullet}} \{m\}) = \{nm + n\} = \{\#\mathsf{mult}\} \mathbin{\bar{\bullet}} \{n\} \mathbin{\bar{\bullet}} (\{\#\mathsf{plus}\} \mathbin{\bar{\bullet}} \{n\} \mathbin{\bar{\bullet}} \{m\})$.

(NzNat Sort). We have $\#\mathsf{NzNat} \in \#\mathsf{inh} \bullet \#\mathsf{Sort}$ by definition.

(NaNat). We have $\#\mathsf{inh} \bullet \#\mathsf{NzNat} = \mathbb{N}_{>0} = \{\#\mathsf{zsucc}\} \mathbin{\bar{\bullet}} \mathbb{N}$.

Recall that we use $\mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^* = \#\mathsf{inh} \bullet \#\mathsf{Sort}$ to denote the set of sort names in M; see Definition D.3(6) and Definition D.5(2).

(Pair Sort). For any $s_1, s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$, we have $s_1 \otimes s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ by definition.

(Pair). For any $s_1, s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ and $x_i \in M_{s_i}$ for $i \in \{1, 2\}$, we have $(x_1, x_2) \in M_{s_1 \otimes s_2} = M_{s_1} \times M_{s_2}$, by definition.

(Pair Fst). For any $s_1, s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ and $x_i \in M_{s_i}$ for $i \in \{1, 2\}$, we have $\#\mathsf{fst} \bullet (x_1, x_2) = \{x_1\}$ by definition.

(Pair Snd). For any $s_1, s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ and $x_i \in M_{s_i}$ for $i \in \{1, 2\}$, we have $\#\mathsf{snd} \bullet (x_1, x_2) = \{x_2\}$ by definition.

(Pair Inj). For any $s_1, s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ and $x_i, y_i \in M_{s_i}$ for $i \in \{1, 2\}$ such that $(x_1, x_2) = (y_1, y_2)$, we have $x_1 = y_1$ and $x_2 = y_2$.

(Pair Domain). For any $s_1, s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$, we have $M_{s_1 \otimes s_2} = M_{s_1} \times M_{s_2} = \{(x_1, x_2) \mid x_1 \in M_{s_1}, x_2 \in M_{s_2}\}$, by definition.

(Proj First). For any $s_1, s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ and $x_i \in M_{s_i}$ for $i \in \{1, 2\}$, we have $\{\#\mathsf{proj}\} \mathbin{\bar{\bullet}} \{1\} \mathbin{\bar{\bullet}} \{(x_1, x_2)\} = x_1$ by definition.

(Proj Rest). For any $s_1, s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ and $x_i \in M_{s_i}$ for $i \in \{1, 2\}$, and any $n \geq 1$, we have $\{\#\mathsf{proj}\} \mathbin{\bar{\bullet}} \{n + 1\} \mathbin{\bar{\bullet}} \{(x_1, x_2)\} = \{\#\mathsf{proj}\} \mathbin{\bar{\bullet}} \{n\} \mathbin{\bar{\bullet}} \{x_2\}$ if x_2 is a pair (note that tuples are just nested pairs); Otherwise, both equal to $\emptyset$.

(Function Sort). For any $s_1, s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$, we have $s_1 \mathbin{\bigcirc\mkern-14mu\rightarrow} s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ by definition.

(Function Domain). For any $s_1, s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ (where s_1 could be a pair/tuple sort, or the unit sort *Unit*), if $F_{s_1, s_2} \neq \emptyset$ then we have $M_{s_1 \bigcirc\mkern-14mu\rightarrow s_2} = [M_{s_1} \to M_{s_2}]$, by definition, then for each $f \colon M_{s_1} \to M_{s_2}$ and any $x \in M_{s_1}$, we have $f(x) \in M_{s_2}$. On the other hand, if $F_{s_1, s_2} = \emptyset$, then $M_{s_1 \bigcirc\mkern-14mu\rightarrow s_2} = \emptyset$ by definition, and the axiom holds because there are no functions in $M_{s_1 \bigcirc\mkern-14mu\rightarrow s_2}$.

(Function Ext). For any $s_1, s_2 \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ and $f, g \colon M_{s_1} \to M_{s_2}$ (note that this implies that $F_{s_1, s_2} \neq \emptyset$), we have $f = g$ iff f and g have the same extension, by definition.

(Unit Sort). We have $\#\mathsf{Unit} \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ by definition.

(Unit). We have $\#\mathsf{unit} \in M_{Unit}$ by definition.

(Unit Domain). We have $M_{Unit} = \{\#\mathsf{unit}\}$ by definition.

(Unit Identity). For any $s \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ and $x \in M_s$, we have $x \bullet \#\mathsf{unit} = x$ by definition.

(Sort). We have $\#\mathsf{s} \in \mathbb{S}\mathrm{ort}\mathbb{N}\mathrm{ame}^*$ for each $s \in S$, by definition.

(Function). We have $f_M = f_A\colon M_{s_1} \times \cdots \times M_{s_n} \to M_s$ for each $f \in F_{s_1\ldots s_n,s}$ by definition.
(Signature Operation). We have $M_{SigOps} = \{f_A \mid f \in F\}$ by definition.
(Signature ArgTuple). We have $M_{SigArgs} = \bigcup_{f\in F_{s_1\ldots s_n,s}} M_{s_1} \times \cdots \times M_{s_n}$ by definition.

D.1.6 Proof of Theorem 6.5

So far, we have completed the construction of the standard model M w.r.t. A. For future reference, let us re-state it in the following definition.

Definition D.6. For a signature (S, F) and an (S, F)-algebra A, we define the corresponding matching logic model $M \vDash \mathsf{ALGEBRA}(S, F)$, whose carrier set, symbol interpretation, and application interpretation are given by Definitions D.3-D.5, respectively.

Lemma D.7. *Under the conditions and notations in Definition D.6, we have $M_s = A_s$ for each sort $s \in S$ and $M_f = A_f$ for each operation $f \in F$.*

Proof. For any sort $s \in S$, we have $M_s = |[\![s]\!]|_M = \#\mathsf{inh} \bullet \#\mathsf{s} = A_s$. For any operation $f \in F_{s_1\ldots s_n,s}$ and all $a_i \in A_{s_i}$, $1 \le i \le n$, we have $f_M \bullet \langle a_1, \cdots, a_n\rangle_M = f \bullet (a_1, \ldots, a_n) = f(a_1, \ldots, a_n)$. Therefore, the derived function $M_f = A_f$ for all operations $f \in F$. □

Theorem 6.5 is then a direct consequence of Lemma D.7.

Theorem 6.5. *Let (S, F) be a signature and A be any (S, F)-algebra. Then there exists a matching logic model $M \vDash \mathsf{Signature}(S, F)$ such that $\alpha(M)$ is exactly A.*

D.2 Properties about $\mathsf{EQSPEC}(E)$ and Proofs of Theorems 6.7 and 6.9

Firstly, we show that M and $A = \alpha(M)$ interpret all terms in the same way. Note that given an A-valuation $\varrho\colon V \to A$, there is an equivalent matching logic valuation ρ such that $\rho(x) = \varrho(x)$ for all $x \in A_{\mathsf{sort}(x)}$. Under these two equivalent valuations, any term $t \in T_F(V)$ is interpreted the same in both models M and $A = \alpha(M)$. Formally,

Lemma D.8. *Under the above conditions about ρ and ϱ, we have $|t|_\rho = \{\bar{\varrho}(t)\}$ for all $t \in T_F(V)$.*

Proof. We apply structural induction on t.
(t is x). We have $|x|_\rho = \{\rho(x)\} = \{\varrho(x)\}$, by the equivalence between ρ and ϱ.
(t is $f(t_1, \ldots, t_n)$). We have $|f(t_1, \ldots, t_n)|_\rho = M_f(|t_1|_\rho, \ldots, |t_n|_\rho) = M_A(\bar{\varrho}(t_1), \ldots, \bar{\varrho}(t_m)) = \bar{\varrho}(f(t_1, \ldots, t_n))$, where the first "=" is by Proposition 5.7 and the rest is by the construction of $\alpha(M)$ (see Definition 6.2). □

Now, we prove Theorem 6.7.

Theorem 6.7. *Let $M \vDash \mathsf{EQSPEC}(F, E)$ and $\alpha(M)$ be the derived F-algebra. Then, for any F-equation e, we have $M \vDash e$ iff $\alpha(M) \vDash_{\mathsf{Alg}} e$. Particularly, we know that $\alpha(M)$ is an (F, E)-algebra.*

Proof. Let e be any F-equation $\forall V.\, t = t'$, where $V = \{x_1, \ldots, x_n\}$ is a set of sorted variables. According to Definition 6.6, equation $\forall V.\, t = t'$ is translated to matching logic pattern using sorted quantification: $\forall x_1{:}\mathsf{sort}(x_1). \ldots x_n{:}\mathsf{sort}(x_n).\, t = t'$. Thus, only matching logic valuations that map the variables $x_1, \ldots, x_n$ to their intended sorts matter. Thus, we have the following reasoning:

$$\begin{aligned} M \vDash \forall V.\, t = t' \quad &\text{iff} \quad |t|_\rho = |t'|_\rho \text{ for all } M\text{-valuation } \rho \text{ such that } \rho(x_i) \in M_{s_i},\ 1 \le i \le n \\ &\text{iff} \quad \bar{\varrho}(t) = \bar{\varrho}(t') \text{ for all } \alpha(M)\text{-valuation } \varrho \\ &\text{iff} \quad \alpha(M) \vDash_{\mathsf{Alg}} \forall V.\, t = t'. \end{aligned}$$

According to specification $\mathsf{EQSPEC}(F, E)$, we have $M \vDash e$ for all $e \in E$. Therefore, $\alpha(M) \vDash_{\mathsf{Alg}} e$ for all $e \in E$, and thus $\alpha(M)$ is an (F, E)-algebra. □

D.3 Equational Specifications in Matching Logic: An Institutional View

The theory of institutions [Goguen and Burstall, 1992] is a category-based theory that formalizes the intuitive notion of a logical system, including syntax, semantics, and the satisfaction relation between them. Here we use the theory of institutions to formalize the relationship between equational specifications (F, E) and matching logic specifications $\mathsf{EQSPEC}(F, E)$. In Appendix D.3.1, we review the basic concepts and definitions about institutions. In Appendix D.3.2, we prove that matching logic forms an institution. In Appendix D.3.3, we show that there is a theoroidal comorphism from equational specifications to matching logic.

D.3.1 Institutions, Institution Morphisms, and Institution Comorphisms

We follow the definitions in [Diaconescu, 2008; Goguen and Rosu, 2002].

Definition D.9. An *institution* $\mathbb{I} = (\mathbf{Sign}, \mathbf{Mod}, \mathbf{Sen}, \models)$ consists of a category $\mathbf{Sign}$ whose objects are called *signatures*, a functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$ giving for each signature Σ a category of Σ-*models*, a functor $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$ giving for each signature a set of Σ-*sentences*, and a Σ-indexed relation $\models\, = \{\models_\Sigma \mid \Sigma \in \mathbf{Sign}\}$ with $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$, such that for any signature morphism $\xi\colon \Sigma \to \Sigma'$, the following *satisfaction relation*:

$$M' \models_{\Sigma'} \mathbf{Sen}(\xi)(\varphi) \quad \text{if and only if} \quad \mathbf{Mod}(\xi)(M') \models_\Sigma \varphi$$

holds for all $M' \in |\mathbf{Mod}(\Sigma')|$ and $\varphi \in \mathbf{Sen}(\Sigma)$.

We use the following diagram to illustrate the satisfaction relation:

$$\begin{tikzcd}
\Sigma \arrow[d, "\xi"] & {|\mathbf{Mod}(\Sigma)|} \arrow[r, dash, "\models_\Sigma"] & \mathbf{Sen}(\Sigma) \arrow[d, "\mathbf{Sen}(\xi)"] \\
\Sigma' & {|\mathbf{Mod}(\Sigma')|} \arrow[u, "\mathbf{Mod}(\xi)"] \arrow[r, dash, "\models_{\Sigma'}"] & \mathbf{Sen}(\Sigma')
\end{tikzcd}$$

Definition D.10. Given an institution $\mathbb{I} = (\mathbf{Sign}, \mathbf{Mod}, \mathbf{Sen}, \models)$, we define its *theoroidal institution* as $\mathbb{I}^{\mathsf{th}} = (\mathbf{Th}, \mathbf{Mod}^{\mathsf{th}}, \mathbf{Sen}^{\mathsf{th}}, \models^{\mathsf{th}})$, where $\mathbf{Th}$ is the category of theories of $\mathbb{I}$, $\mathbf{Mod}^{\mathsf{th}}$ is the extension of $\mathbf{Mod}$ to theories, $\mathbf{Sen}^{\mathsf{th}}$ is $\mathbf{sign}; \mathbf{Sen}$, and $\models^{\mathsf{th}}$ is $\mathbf{sign}; \models$, where $\mathbf{sign} : \mathbf{Th} \to \mathbf{Sign}$ is the functor that forgets the sentences of a theory.

The institutional mechanism for expressing encodings of logics is that of *comorphism*.

Definition D.11. A *comorphism of institutions* $(\Phi, \alpha, \beta) : \mathbb{I} \to \mathbb{I}'$, where $\mathbb{I} = (\mathbf{Sign}, \mathbf{Mod}, \mathbf{Sen}, \models)$ and $\mathbb{I}' = (\mathbf{Sign}', \mathbf{Mod}', \mathbf{Sen}', \models')$, consists of a functor $\Phi : \mathbf{Sign} \to \mathbf{Sign}'$, a natural transformation $\beta : \Phi^{op}; \mathbf{Mod}' \Rightarrow \mathbf{Mod}$, and a natural transformation $\alpha : \mathbf{Sen} \Rightarrow \Phi; \mathbf{Sen}'$ such that the following *satisfaction condition*:

$$\beta_\Sigma(M') \models_\Sigma e \quad \text{if and only if} \quad M' \models_{\Phi(\Sigma)} \alpha_\Sigma(e)$$

holds for any $\Sigma \in |\mathbf{Sign}|$, $M' \in \left|\mathbf{Mod}'(\Phi(\Sigma))\right|$, and $e \in \mathbf{Sen}(\Sigma)$.

We use the following diagram to illustrate the satisfaction condition:

$$\begin{tikzcd}
\Sigma \arrow[d, "\Phi"] & {|\mathbf{Mod}(\Sigma)|} \arrow[r, dash, "\models_\Sigma"] & \mathbf{Sen}(\Sigma) \arrow[d, "\alpha_\Sigma"] \\
\Phi(\Sigma) & {|\mathbf{Mod}(\Phi(\Sigma))|} \arrow[u, "\beta_\Sigma"] \arrow[r, dash, "\models_{\Sigma'}"] & \mathbf{Sen}(\Phi(\Sigma))
\end{tikzcd}$$

Definition D.12. We make the following definitions.

1. A functor $\Phi : \mathbf{Th} \to \mathbf{Th}'$ is *signature preserving* iff there is a functor $\Phi^\diamond : \mathbf{Sign} \to \mathbf{Sign}'$ such that $\Phi; \mathbf{sign}' = \mathbf{sign}; \Phi^\diamond$.
2. A *theoroidal comorphism* is a comorphism $(\Phi, \alpha, \beta) : \mathbb{I}^{\mathsf{th}} \to \mathbb{I}'^{\mathsf{th}}$ with Φ signature preserving.
3. A *simple theoroidal comorphism* is a comorphism $(\Phi, \alpha, \beta) : \mathbb{I} \to \mathbb{I}'^{\mathsf{th}}$ with Φ signature preserving.

D.3.2 The Institution of Matching Logic

Recall that a matching logic signature Σ is simply a set of (constant) symbols. Then, for any two (nonempty) matching logic signatures Σ and Σ', a *matching logic signature morphism* $\xi\colon \Sigma \to \Sigma'$ is a function from Σ to Σ'. Therefore, the set of nonempty matching logic signatures forms a category, denoted $\mathbf{Sig}_{\mathsf{ML}}$, where objects are nonempty signatures and morphisms are functions between signatures.

Remark D.13. We only consider nonempty matching logic signatures in this section.

Definition D.14. Let Σ be a matching logic signature. A Σ-*morphism* from a Σ-model M to another M' is a function $f\colon M \to M'$ such that

1. for all $a_1, a_2 \in M$, $\bar{f}(a_1 \bullet_M a_2) = f(a_1) \bullet_{M'} f(a_2)$;
2. for all $\sigma \in \Sigma$, $\bar{f}(\sigma_M) = \sigma_{M'}$;

where $\_\bullet_M\_$ is the application interpretation in M; $\_\bullet_{M'}\_$ is the application interpretation in M'; and $\bar{f}\colon \mathcal{P}(M) \to \mathcal{P}(M')$ is the pointwise extension of f, defined as $\bar{f}(A) = \{f(a) \mid a \in A\}$ for all $A \subseteq M$.

Lemma D.15. *Let Σ be a matching logic signature. Then Σ-models form a category $\mathbf{Mod}_{\mathsf{ML},\Sigma}$, whose objects are Σ-models and morphisms are Σ-morphisms.*

Proof. We show the existence of identity and composition, and prove their properties.

(Existence of identity). Let M be any Σ-model. Let $id_M : M \to M$ be the identity function of a given Σ-model M. Then, id_M is a Σ-morphism, which we call the identity morphism of M.

(Existence of composition). Let $f\colon M \to M'$ and $g\colon M' \to M''$ be two Σ-morphisms. Let $f;g\colon M \to M''$ be the composite of f and g as functions. Then we have $\overline{f;g}(A) = \{g(f(a)) \mid a \in A\}$ for any $A \subseteq M$. We have the following reasoning:

Firstly, for all $a_1, a_2 \in M$,

$$\begin{aligned}
\overline{f;g}(a_1 \bullet_M a_2) &= \{g(f(a)) \mid a \in (a_1 \bullet_M a_2)\} \\
&= \{g(b) \mid b \in \bar{f}(a_1 \bullet_M a_2)\} \\
&= \{g(b) \mid b \in f(a_1) \bullet_{M'} f(a_2)\} \\
&= \bar{g}(\bar{f}(a_1) \bullet_{M'} \bar{f}(a_2)) \\
&= g(f(a_1)) \bullet_{M''} g(f(a_2)).
\end{aligned}$$

Secondly, for all $\sigma \in \Sigma$,

$$\begin{aligned}
\overline{f;g}(\sigma_M) &= \{g(f(a)) \mid a \in \sigma_M\} \\
&= \{g(b) \mid b \in \bar{f}(\sigma_M)\} \\
&= \{g(b) \mid b \in \sigma_{M'}\} \\
&= \bar{g}(\sigma_{M'}) \\
&= \sigma_{M''}.
\end{aligned}$$

Hence, $f;g\colon M \to M''$ is a Σ-morphism.

(id_M is identity). This is trivial, due to the properties of the identity functions.

(Composition is associative). This is also trivial, because function composition is associative.

Hence, $\mathbf{Mod}_{\mathsf{ML},\Sigma}$ forms a category. □

Definition D.16. We define the functor $\mathbf{Mod}_{\mathsf{ML}}\colon \mathbf{Sig}_{\mathsf{ML}}^{op} \to \mathbf{Cat}$, which sends every matching logic signature Σ to the category $\mathbf{Mod}_{\mathsf{ML},\Sigma}$ of Σ-models, and sends every matching logic signature morphism $\xi\colon \Sigma \to \Sigma'$ to the functor $\mathbf{Mod}_{\mathsf{ML}}(\xi)\colon \mathbf{Mod}_{\mathsf{ML},\Sigma'} \to \mathbf{Mod}_{\mathsf{ML},\Sigma}$ that:

1. sends a Σ'-model $(M', \_\bullet_{M'}\_, \{\sigma_{M'}\}_{\sigma\in\Sigma})$ to the Σ-model $(M, \_\bullet_M\_, \{\sigma_M\}_{\sigma\in\Sigma})$ with $M = M'$, $\_\bullet_M\_ = \_\bullet_{M'}\_$, and $\sigma_M = (\xi(\sigma))_{M'}$; and

2. sends a Σ'-morphism $f' \colon M_1' \to M_2'$ to the Σ-morphism:

$$\mathbf{Mod}_{\mathsf{ML}}(\xi)(f') = f \colon \mathbf{Mod}_{\mathsf{ML}}(\xi)(M_1') \to \mathbf{Mod}_{\mathsf{ML}}(\xi)(M_2') \qquad \text{defined by } f = f'$$

Definition D.17. We define the functor $\mathbf{Sen}_{\mathsf{ML}} : \mathbf{Sig}_{\mathsf{ML}} \to \mathbf{Set}$, which sends

1. a matching logic signature Σ to $\mathbf{Sen}_{\mathsf{ML}}(\Sigma) = \textsc{Pattern}_\Sigma$, and
2. a matching logic signature morphism $\xi : \Sigma \to \Sigma'$ to its free extension $\bar{\xi} : \textsc{Pattern}_\Sigma \to \textsc{Pattern}_{\Sigma'}$,

where $\textsc{Pattern}_\Sigma$ (resp. $\textsc{Pattern}_{\Sigma'}$) is the set of Σ-patterns (resp. Σ'-patterns).

Proposition D.18. *Let $\xi : \Sigma \to \Sigma'$ be a matching logic signature morphism and M' be a Σ'-model. Then we have,*

$$M' \vDash_{\Sigma'} \mathbf{Sen}_{\mathsf{ML}}(\xi)(\varphi) \quad \textit{if and only if} \quad \mathbf{Mod}_{\mathsf{ML}}(\xi)(M') \vDash_\Sigma \varphi$$

for all $M' \in |\mathbf{Mod}_{\mathsf{ML}}(\Sigma')|$ and $\varphi \in \mathbf{Sen}(\Sigma)$.

Proof. By a mechanical check of the condition from Definition 4.7 using the inductive definition of a valuation given in Definition 4.4. □

Corollary D.19. $\mathbb{ML} = (\mathbf{Sig}_{\mathsf{ML}}, \mathbf{Mod}_{\mathsf{ML}}, \mathbf{Sen}_{\mathsf{ML}}, \vDash)$ *forms the institution of matching logic.*

D.3.3 Theoroidal Comorphisms from Algebras to Matching Logic

Firstly, we recall the classical result that (many-sorted) algebras defined in Definition 3.3 form an institution $\mathbb{ALG}$; see [Goguen and Burstall, 1992]. The institution $\mathbb{ALG}$ consists of:

1. $\mathbf{Sign}_{\mathsf{ALG}}$, which is the category of (many-sorted) signatures;
2. $\mathbf{Mod}_{\mathsf{ALG}}$, which sends each signature (S, F) to the category of (S, F)-algebras;
3. $\mathbf{Sen}_{\mathsf{ALG}}$, which sends each signature (S, F) to the set of (S, F)-equations; and
4. the satisfaction relation $\vDash_{\mathsf{Alg}}$, which relates the (S, F)-algebras with their valid (S, F)-equations.

Theorem D.20. *There exists a simple theoroidal comorphism $(\Phi, \gamma, \delta) : \mathbb{ALG} \to \mathbb{ML}^{\mathsf{th}}$.*

Proof. Let $\Phi \colon \mathbf{Sign}_{\mathsf{ALG}} \to \mathbf{Th}_{\mathsf{ML}}$ be the functor that sends each signature (S, F) to the matching logic specification $\mathsf{ALGEBRA}(S, F)$ according to Definition 6.1, and sends each signature morphism $\xi \colon (S, F) \to (S', F')$ to the matching logic signature morphism

$$\Phi(\xi) \colon \mathsf{ALGEBRA}(S, F) \to \mathsf{ALGEBRA}(S', F'),$$

which renames the sorts and operations symbols according to ξ and keeps the other symbols unchanged. Then, Φ is signature preserving because its restriction to signatures $\Phi^\diamond : \mathbf{Sign}_{\mathsf{ALG}} \to \mathbf{Sig}_{\mathsf{ML}}$ sends a signature (S, F) to the matching logic signature of $\mathsf{ALGEBRA}(S, F)$.

The natural transformation $\delta \colon \Phi^{op}; \mathbf{Mod}_{\mathsf{ML}} \Rightarrow \mathbf{Mod}_{\mathsf{ALG}}$ yields a functor

$$\delta_{(S,F)} \colon \mathbf{Mod}_{\mathsf{ML}}(\mathsf{ALGEBRA}(S, F)) \to \mathbf{Mod}_{\mathsf{ALG}}(S, F)$$

defined by $\delta_{(S,F)}(M') = \alpha(M')$, where α is the model transformation defined in Definition 6.2.

The natural transformation $\gamma : \mathbf{Sen}_{\mathsf{ALG}} \Rightarrow \Phi; \mathbf{Sen}_{\mathsf{ML}}$ yields a function

$$\gamma_{(S,F)} : \mathbf{Sen}_{\mathsf{ALG}}(S, F) \to \mathbf{Sen}_{\mathsf{ML}}(\mathsf{ALGEBRA}(S, F))$$

define by $\gamma_{(S,F)}(e) = e$, according to Definition 6.6.

Then, for any matching logic model $M' \vDash \mathsf{ALGEBRA}(S, F)$ and equation e, the satisfaction condition

$$\delta_{(S,F)}(M') \vDash_{\mathsf{Alg}} e \quad \text{if and only if} \quad M' \models \gamma_{(S,F)}(e)$$

is equivalent to the following statement:

$$\alpha(M') \vDash_{\mathsf{Alg}} e \quad \text{if and only if} \quad M' \models e$$

which then holds by Theorem 6.7. Hence, $(\Phi, \gamma, \delta) : \mathbb{ALG} \to \mathbb{ML}^{\mathsf{th}}$ is a simple theoroidal comorphism. □

Theorem D.21. *There exists a theoroidal comorphism* $(\Phi, \gamma, \delta) : \mathbb{ALG}^{\mathsf{th}} \to \mathbb{ML}^{\mathsf{th}}$.

Proof. We define $\Phi : \mathbf{Th}_{\mathsf{ALG}} \to \mathbf{Th}_{\mathsf{ML}}$ to be the functor that sends each equational specification (S, F, E) to the matching logic specification $\mathsf{EQSPEC}(S, F, E)$, defined in Definition 6.6. The rest of the proof is the same as the proof of Theorem D.20. □

Definition D.22. We define the institution $\mathbb{IAS} = (\mathbf{Th}_{\mathsf{IAS}}, \mathbf{Mod}_{\mathsf{IAS}}, \mathbf{Sen}_{\mathsf{IAS}}, \vDash_{\mathsf{Alg}})$, where:

1. $\mathbf{Th}_{\mathsf{IAS}}$ is the subcategory of $\mathbf{Th}_{\mathsf{ALG}}$ induced by the morphisms that protect the initial models; that is, if $\phi : (S, F, E) \to (S', F', E')$ is a morphism in $\mathbf{Th}_{\mathsf{ALG}}$ and M' is an initial (S', F', E')-model, then $M'|_\phi$ is an initial (S, F, E)-model;

2. $\mathbf{Mod}_{\mathsf{IAS}}$ sends (S, F, E) to the subcategory of the initial (S, F, E)-algebras and sends each $\phi : (S, F, E) \to (S', F', E')$ to $\mathbf{Mod}_{\mathsf{IAS}}(\phi) : \mathbf{Mod}_{\mathsf{IAS}}(S', F', E') \to \mathbf{Mod}_{\mathsf{IAS}}(S, F, E)$, defined by $\mathbf{Mod}_{\mathsf{IAS}}(\phi)(M') = M'|_\phi$;

3. $\mathbf{Sen}_{\mathsf{IAS}}$ sends (S, F, E) to the set of F-equations.

Theorem D.23. *There exists a simple theoroidal comorphism* $(\Phi, \gamma, \delta) : \mathbb{IAS} \to \mathbb{ML}^{\mathsf{th}}$.

Proof. We let $\Phi : \mathbf{Th}_{\mathsf{IAS}} \to \mathbf{Th}_{\mathsf{ML}}$ to be the functor that sends each (S, F, E) to $\mathsf{EQSPEC}(S, F, E)$, according to Definition 6.6. We define $\delta(S, F, E)$, which sends a model $M' \vDash \mathsf{EQSPEC}(S, F, E)$ to the derived algebra $\beta(M')$, where the model transformation β is defined in Theorem 8.3. We define $\gamma_{(S,F,E)}$ like in the proof of Theorem D.20. The rest of the proof is the same as the proof of Theorem D.20. □

E Proofs of the Results in Section 7

Lemma 7.3. *For any $M \vDash \mathsf{NOCONFUSION}(F)$, $\alpha(M)$ satisfies no-confusion. Particularly, for any $t, t' \in T_F$, these are equivalent: (1) $M \vDash t = t'$; (2) $\alpha(M) \vDash_{\mathsf{Alg}} \forall \emptyset .\, t = t'$; (3) t and t' are the same.*

Proof. Note that (1) and (2) are equivalent due to Theorem 6.7. We only need to prove that (1) and (3) are equivalent. Clearly, (3) implies (1), so we only need to prove that (1) implies (3).

The proof is based on structural induction. Let t and t' be two ground terms $f(t_1, \ldots, t_n)$ and $f'(t'_1, \ldots, t'_{n'})$, where $n, n' \geq 0$. If $M \vDash f(t_1, \ldots, t_n) = f'(t'_1, \ldots, t'_{n'})$, then by axiom (No Confusion), we have $M \vDash f = f'$ and $M \vDash \langle t_1, \ldots, t_n \rangle = \langle t'_1, \ldots, t'_{n'} \rangle$. For the former, we conclude that f and f' are the same, according to axiom (Distinct Function). For the latter, we conclude by axiom (Pair Inj) that $n = n'$ and that $M \vDash t_1 = t'_1$, $\ldots$, $M \vDash t_n = t'_n$. Therefore, by induction hypothesis, we know that t_i and t'_i are the same term, and thus $f(t_1, \ldots, t_n)$ and $f'(t'_1, \ldots, t'_{n'})$ are the same term. □

Lemma 7.8. *For $M \vDash \mathsf{NOJUNK}(S, F)$, the derived algebra $A = \alpha(M)$ satisfies no-junk. Specifically, for any $s \in S$ and $a \in A_s$, there exists a ground term $t_a \in T_{F,s}$ such that $\mathsf{eval}(t_a) = a$ (see Remark 3.6).*

Proof. Note that for any void sort, its carrier set is empty according to axiom (No Junk Void), which satisfies no-junk because it has no ground terms. Therefore, we only need to consider non-void sorts.

Our proof uses [Meseguer and Goguen, 1985, Proposition 15], which states that no-junk holds iff the (unique) morphism $h\colon T_F \to A$ is surjective. Recall that M_s for $s \in S$ is the carrier set of s in M. For notational brevity, we define $\langle M\rangle = M_{s_1} \times \cdots \times M_{s_n}$. Then by axiom (No Junk Non-Void), we have $\langle M\rangle = \mu D.\, \mathcal{F}$, where the function $\mathcal{F}$ is the function that maps set D to:

$$\mathcal{F}(D) = \{M_f(a_1, \ldots, a_m) \mid f \in F_{t_1 \ldots t_m, s_1}, a_i \in D_{t_i}\} \times \cdots \times \{M_f(a_1, \ldots, a_m) \mid f \in F_{t_1 \ldots t_m, s_n}, a_i \in D_{t_i}\}$$

where $D_t = \mathit{proj}_M \bullet \{i\} \bullet D$ if t is a non-void sort $s_i \in S_{\mathsf{nonvoid}}$, or $D_t = \emptyset$ if t is a void sort. Note that if t is void, then $D_t = \emptyset$, so it does not affect the definition of $\mathcal{F}$ and can be omitted. In the following, we safely assume that all appearing sorts are non-void.

By definition, $\langle M\rangle$ is the smallest (product) set D such that $D = \mathcal{F}(D)$, where $D = D_1 \times \cdots \times D_n$. Let us consider a new set $D' = D'_1 \times \cdots \times D'_n$, defined by $D'_i = \{\mathsf{eval}(t) \mid t \in T_{F,s_i}\}$ for each $1 \le i \le n$, which includes precisely all *reachable elements*. Clearly, D' is a fixpoint of $\mathcal{F}$ due to the inductive definition of terms, and thus $\langle M\rangle \subseteq D'$ due to the least fixpoint pattern semantics. Since $\langle M\rangle$ is the carrier set(s) of the derived algebra $A = \alpha(M)$ (Definition 6.2), we know that the morphism $h\colon T_F \to A$ is indeed surjective. Hence, by [Meseguer and Goguen, 1985, Proposition 15], the derived algebra $\alpha(M)$ satisfies no-junk. □

Theorem 7.9. *For $M \vDash \mathsf{TERMALGEBRA}(F)$, the derived algebra $\alpha(M)$ is the term algebra T_F.*

Proof. We have proved that $\alpha(M)$ satisfies both no-confusion and no-junk. By Theorem 3.22, $\alpha(M)$ is the term algebra. □

F Proofs of the Results in Section 8

Theorem 8.2. *For any $M \vDash \mathsf{INITIALALGEBRA}(S, F, E)$, the derived model $\alpha(M)$ is exactly the term algebra T_F (Theorem 7.9). Then, the interpretation Eq_M is a binary relation over ground terms in T_F, and we have that $(t, t') \in \mathit{Eq}_M$ iff $t \simeq_E t'$ for all $t, t' \in T_F$, where $\simeq_E$ is defined in Proposition 3.15.*

Proof. Firstly, we show that the equivalence relation $\simeq_E$ in the term algebra T_F is the smallest relation generated by equational deduction (Definition 3.10), where the substitution rule is not needed because all terms are ground terms without variables. Thus, $\simeq_E$ is defined as the smallest set that includes the identity relation and all (ground) instances of the equations in E, and is closed under symmetry, commutativity, associativity, and congruence.

By axiom (Equivalence), we know that Eq_M, the interpretation of Eq in M that forms a binary relation on T_F, includes the identity relation and is closed under symmetry, commutativity, associativity, and congruence. Therefore, we only need to prove that pattern $\bigvee_{(\forall V.\, t=t') \in E} \exists V.\, \langle t, t'\rangle$ includes all ground instances of the equations in E.

Let $(\forall V.\, t = t') \in E$ with $V = \{x_1, \ldots, x_n\}$ and $\forall \emptyset.\, u = u'$ be one of its ground instances, where $u = t[t_1/x_1 \ldots t_n/x_n]$ and $u' = t'[t_1/x_1 \ldots t_n/x_n]$ for $t_1, t'_1 \in T_{F,\mathsf{sort}(x_1)}, \ldots, t_n, t'_n \in T_{F,\mathsf{sort}(x_n)}$. By standard FOL reasoning, we have $\mathsf{INITIALALGEBRA}(F, E) \vdash \langle u, u'\rangle \to \exists V.\, \langle t, t'\rangle$, so all ground instances are included in $\bigvee_{(\forall V.\, t=t') \in E} \exists V.\, \langle t, t'\rangle$. Since the derived model $\alpha(M)$ is the term algebra, quantification $\exists V$ ranges only the (ground) terms of the appropriate sorts. We conclude that $\bigvee_{(\forall V.\, t=t') \in E} \exists V.\, \langle t, t'\rangle$ indeed includes the ground instances of the equations in E. Thus, Eq_M and $\simeq_M$ are both the smallest binary relation that includes identity and ground instances of E, and is closed under symmetry, commutativity, associativity, and congruence, and therefore they are the same. Hence, we have $(t, t') \in \mathit{Eq}_M$ iff $t \simeq_E t'$. □

Theorem 8.3. *Under the conditions and notations in Theorem 8.2, we define $\beta(M)$ as the Eq_M-quotient algebra of $\alpha(M)$. Then, $\beta(M)$ is exactly the quotient term algebra $T_{F/E}$.*

Proof. Since $\alpha(M)$ is the term algebra T_F and Eq_M is the equivalence relation $\simeq_M$, the quotient algebra $\beta(M)$ is by definition the quotient term algebra $T_{F/E}$, according to Theorem 3.19. □

G Proof of Theorem 9.1

Theorem 9.1. $\mathsf{INITIALALGEBRA}(E^{Nat}) \vdash \underbrace{(\mu D.\, \mathit{zero} \vee \mathit{succ}(D) \vee \mathit{plus}(D, D))}_{\textit{equals to } [\![Nat]\!] \textit{ by axiom } \text{(No Junk)}} \simeq (\mu D.\, \mathit{zero} \vee \mathit{succ}(D))$.

Proof. Let us denote the LHS as Φ_1 and the RHS as Φ_2. We need to prove $\vdash \Phi_1 \subsetneq \Phi_2$ and $\vdash \Phi_2 \subsetneq \Phi_1$. The latter follows directly from definition, so we only prove the former.

Let us define $[x] \equiv \exists y.\, y \wedge x \simeq y$ that is matched by all elements y that is E-equivalent to x, and its pointwise extension $[\varphi] \equiv \exists x.\, [x] \wedge x \in \varphi$. Then, we need to prove $\vdash [\Phi_1] \to [\Phi_2]$, where we transform $\subsetneq$ to $\to$, so as to apply (Knaster-Tarski). By matching logic reasoning, we transform the proof goal to $\vdash \Phi_1 \to \exists z.\, z \wedge [z] \subseteq [\Phi_2]$, which is equivalent to $\vdash \Phi_1 \to \exists z.\, z \wedge z \in [\Phi_2]$. By (Knaster-Tarski), we have three sub-goals:

$$\begin{aligned}
&\vdash \mathit{zero} \to \exists z.\, z \wedge z \in [\Phi_2] \\
&\vdash z \in [\Phi_2] \to \mathit{succ}(z) \in [\Phi_2] \\
&\vdash z_1 \in [\Phi_2] \wedge z_2 \in [\Phi_2] \to \mathit{plus}(z_1, z_2) \in [\Phi_2]
\end{aligned}$$

The first two follow by the definition of Φ_2. For the last one, we transform it to $\vdash [\Phi_2] \to \exists z_2.\, z_2 \wedge \forall z_1.\, z_1 \in [\Phi_2] \subseteq \mathit{plus}(z_1, z_2) \in [\Phi_2]$, and denote its RHS as Φ_3. To prove $\vdash [\Phi_2] \to \Phi_3$, we need to prove $\vdash \Phi_2 \to \exists w.\, w \wedge [w] \subseteq \Phi_3$. By (Knaster-Tarski), we have two sub-goals:

$$\begin{aligned}
&\vdash [\mathit{zero}] \subseteq \Phi_3 \\
&\vdash [w] \subseteq \Phi_3 \to [\mathit{succ}(w)] \subseteq \Phi_3
\end{aligned}$$

For the first one, we prove $\vdash \forall x.\, x \simeq \mathit{zero} \to \forall z_1.\, z_1 \in [\Phi_2] \subseteq \mathit{plus}(z_1, x) \in [\Phi_2]$. By the Peano axioms, we have $\mathit{plus}(z_1, \mathit{zero}) \simeq z_1$, so we only need to prove $\vdash \forall z_1.\, z_1 \in [\Phi_2] \to z_1 \in [\Phi_2]$, which clearly holds. For the second one, we prove $\vdash [w] \subseteq \Phi_3 \wedge \forall x.\, x \simeq \mathit{succ}(w) \to \forall z_1.\, z_1 \in [\Phi_2] \subseteq \mathit{plus}(z_1, x) \in [\Phi_2]$, which is equivalent to $\vdash [w] \subseteq \Phi_3 \to \forall z_1.\, z_1 \in [\Phi_2] \subseteq \mathit{plus}(z_1, \mathit{succ}(w)) \in [\Phi_2]$. By the Peano axioms, we have $\mathit{plus}(z_1, \mathit{succ}(w)) \simeq \mathit{succ}(\mathit{plus}(z_1, w))$, so we only need to prove $\vdash [w] \subseteq \Phi_3 \to \forall z_1.\, z_1 \in [\Phi_2] \subseteq \mathit{succ}(\mathit{plus}(z_1, w)) \in [\Phi_2]$, which holds by definition of Φ_2. $\square$

H Defining Order-Sorted Initial Algebras in Matching Logic

In this section, we define order-sorted algebras (OSA) in matching logic, where subsorting and operation overloading are defined axiomatically by patterns. We follow the definitions and notations in [Goguen and Meseguer, 1992]. We use $(S, \leq)$ to denote the sort set and the subsorting partial ordering $\leq \subseteq S \times S$. We write $s_1 \ldots s_n \leq s'_1 \ldots s'_n$ to mean $s_1 \leq s'_1, \ldots, s_n \leq s'_n$. We require the following conditions on an order-sorted signature $(S, \leq, F)$:

1. (monotonicity). If $F_{s_1 \ldots s_n, s} \cap F_{s'_1 \ldots s'_n, s'} \neq \emptyset$, and $s_i \leq s'_i$ for $1 \leq i \leq n$, then $s \leq s'$;
2. (regularity). For any $f \in F_{s'_1 \ldots s'_n, s'}$ and $s''_i \leq s'_i$ for $1 \leq i \leq n$, there exists the least sort sequence $s_1, \ldots, s_n, s$ (w.r.t. $\leq$), such that $s''_i \leq s_i$ for $1 \leq i \leq n$, and $f \in F_{s_1 \ldots s_n, s}$.

We only consider signatures that are monotone and regular.

Definition H.1. Let $(S, \leq, F)$ be an order-sorted signature. An $(S, \leq, F)$-algebra A consists of:

1. a family of carrier sets A_s for each $s \in S$; and
2. an operation interpretation $A_f \colon A_{s_1} \times \cdots \times A_{s_n} \to A_s$ for every $f \in F_{s_1 \ldots s_n, s}$;

such that:

1. $s \leq s'$ implies $A_s \subseteq A_{s'}$; and

2. $f \in F_{w_1,s_1} \cap F_{w_2,s_2}$ and $w_1 \leq w_2$ imply that $A_f : A_{w_1} \to A_{s_1}$ equals to $A_f : A_{w_2} \to A_{s_2}$ on A_{w_1}.

Now we define the matching logic specification $\mathsf{TERMALGEBRA}(S, \leq, F)$ that captures the order-sorted term algebra T_F, by extending the specification $\mathsf{TERMALGEBRA}(S, F)$ of many-sorted term algebras, defined in Section 7.3, with the subsorting axioms:

$$\text{(Subsorting)} \qquad [\![s]\!] \subseteq [\![s']\!] \text{ for } s \leq s'$$

Note that operation overloading is obtained by defining one (Function) axiom for each instance/copy of the operation $f \in F$. In the following, we can extend the model transformation α in Definition 6.2 to order-sorted algebras as follows:

Proposition H.2. *For any $M \vDash \mathsf{TERMALGEBRA}(S, \leq, F)$, we define M_s as the carrier set of s in M, and define $M_f : A_{s_1} \times \cdots \times A_{s_n} \to A_s$ for $f \in F_{s_1 \ldots s_n, s}$. Then, by* (Subsorting)*, $M_s \subseteq M_{s'}$ for all $s \leq s'$. By construction, $M_f : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ and $M_f : M_{s'_1} \times \cdots \times M_{s'_n} \to M_{s'}$ coincide on $M_{s_1} \times \cdots \times M_{s_n}$, if $s_1 \ldots s_n \leq s'_1 \ldots s'_n$. Hence, the derived algebra $\alpha(M)$ is an order-sorted algebra.*

In addition, we know from Theorem 7.9 that $\alpha(M)$ is the many-sorted term algebra of (S, F) when we ignore subsorting. Therefore, by Proposition H.2, $\alpha(M)$ is the order-sorted term algebra of $(S, \leq, F)$:

Theorem H.3. *Under the conditions and notations in Proposition H.2, $\alpha(M)$ is the order-sorted term algebra T_F.*