Technical Report: Applicative Matching Logic: Semantics of $\mathbb{K}$

Xiaohong Chen and Grigore Roşu
{xc3,grosu}@illinois.edu

March 18, 2020

Abstract

$\mathbb{K}$ is a formal language semantics framework where programming languages are defined using configurations and rewrite rules. Language tools such as interpreters and program verifiers are automatically generated from language definitions. The wide applications of $\mathbb{K}$ to real-world languages raise interest in its logical foundation and formal semantics. Existing work only considers fragments of $\mathbb{K}$ and formalizes a subset of its features. This paper proposes a simple logic called *applicative matching logic* (AML) as a new logical foundation for $\mathbb{K}$. We show that AML captures all the logical frameworks that have been implemented fully or partially in $\mathbb{K}$, including: many-sorted first-order logic (MSFOL); constructors and term algebras, with the inductive principle; MSFOL variants such as algebraic specifications, FOL with partial functions, and FOL with least fixpoints; order-sorted algebras; and parametric sorts/types. We then put everything together and formalize the semantics of $\mathbb{K}$, by showing how program execution and verification are heuristics for AML proof search. Finally, we propose two new features for $\mathbb{K}$, function and dependent sorts/types, based on the solid mathematical foundation provided by AML.

1 Introduction

The $\mathbb{K}$ framework (`www.kframework.org`) [55] is an effort in pursuing the ideal language framework (Fig. 1), where all programming languages must have formal semantic definitions and all language tools are automatically derived in a correct-by-construction manner at no additional cost. $\mathbb{K}$ has been used to develop the complete formal definitions of many real-world languages such as C [30], Java [5], JavaScript [49], x86 [16] as well as emerging blockchain languages such as EVM [32] and IELE [35]. All language tools such as parsers, interpreters, and deductive program verifiers are automatically derived from the formal language definitions.

The wide practical applications of $\mathbb{K}$ have raised research interest in and questions about its own logical foundations. That is: *What is the semantics of* $\mathbb{K}$*?* To answer that, we look for a foundational logic where every $\mathbb{K}$ definition of a language L defines a *logical theory* Γ^L, and different language tools represent different best-effort implementations of logic reasoning within Γ^L. For example, the program verifier is a best effort implementation of the logic reasoning $\Gamma^L \vdash \varphi_{\text{pre}} \Rightarrow \varphi_{\text{post}}$, which intuitively means that all program configurations that satisfy φ_{pre} must reach configurations that satisfy φ_{post}, on termination.

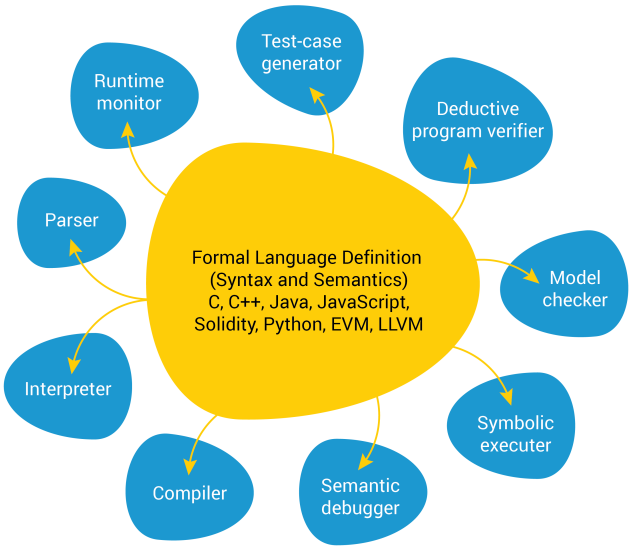

Figure 1: An ideal language framework vision

There is a series of research work [56, 54, 53, 39, 45, 9] that aims at giving a formal semantics to $\mathbb{K}$. The implementation of $\mathbb{K}$ has more than 130,00 lines of code. However, the existing work only considers fragments of $\mathbb{K}$, each formalizing only a subset of its features; see also Section 10. In this paper, we propose a simple logic that can formalize all $\mathbb{K}$'s features.

1.1 A Running Example and Main $\mathbb{K}$ Features

We discuss the main features of $\mathbb{K}$ using the simple, prototypical imperative language IMP as a running example. We show the complete formal $\mathbb{K}$ definition of IMP in Fig. 2. Readers need not understand all details (which can be found in [11]); we explain the relevant parts below.

There are two modules: `IMP-SYNTAX` defines the concrete syntax of IMP, and `IMP` defines program configurations, and formal semantics by rewrite rules. In `IMP-SYNTAX`, $\mathbb{K}$ uses the conventional BNF notation, where nonterminals begin with capital letters and terminals are quoted strings. We call nonterminals *sorts*; e.g., `Exp` is the sort of expressions; `Stmt` is the sort of program statements, etc. Sorts can be *ordered*; e.g., `Exp` takes `Int` and `Id` as its *subsorts* (Fig. 2, line 2), meaning that integers and identifiers (i.e., program variables) are well-formed expressions.

$\mathbb{K}$ has builtin support for *parametric sorts*; e.g., `Ids` is the sort of identifier lists (Fig. 2, line 13) that is defined from `Id` using the generic parametric sort `List`. Intuitively, `List` is a *sort constructor* that takes a base sort (e.g., `Id`) and a delimiter (e.g., `","`), and produces the lists of that base sort. The delimiter is a frontend detail, only relevant for parsing purpose, so it is not of our interest here. Here we are interested in the $\mathbb{K}$ feature of defining parametric sorts such as `List`.

$\mathbb{K}$ uses *configurations* to structure semantic data needed for program execution. Configurations are built from *cells*, possibly nested, written in XML format. The configurations of IMP (lines 18-19)

have two *cells*: a <k/> cell that contains program code to be executed and a <state/> cell that contains program states mapping program variables to values.

𝕂 uses *rewrite rules*, or just *rules*, to define the formal semantics of IMP. Rules have the form $\varphi_{lhs} \Rightarrow \varphi_{rhs}$, where $\varphi_{lhs}, \varphi_{rhs}$ are *symbolic configuration patterns* that can be matched by concrete IMP configurations; e.g., `if(0) S1 S2 => S2` is a rule that rewrites the program code `if(0) S1 S2` in one step to `S2`, as expected. Rules can be conditional; e.g., `if(I) S1 S2 => S1 requires I =/=Int 0` rewrites the program code `if(I) S1 S2` in one step to `S1`, if `I` is nonzero. In this way, the rewrite rules in the module `IMP` define a *transition system* $\mathcal{S}^{\mathsf{IMP}}$ over IMP configurations. Then, executing an IMP program `$PGM` is equivalent to generating a trace of $\mathcal{S}^{\mathsf{IMP}}$ from the initial configuration <k> `$PGM` </k> <state> . </state>, where `$PGM` (line 19) is a special variable that tells 𝕂 to put the program in the <k/> cell, and "." denotes "nothing", i.e., the empty program state. Similarly, various formal program analyses are equivalent to reasoning about all traces from initial configuration patterns, which sometimes can be symbolic.

```
1  module IMP-SYNTAX imports DOMAINS-SYNTAX
2    syntax Exp ::= Int | Id | Exp "+" Exp | Exp "-" Exp
3      | "(" Exp ")" [bracket]
4
5    syntax Stmt ::= Id "=" Exp ";" [strict(2)]
6      | "if" "(" Exp ")" Stmt Stmt [strict(1)]
7      | "while" "(" Exp ")" Stmt
8      | "{" Stmt "}" [bracket]
9      | "{" "}"
10     > Stmt Stmt [left]
11
12   syntax Pgm ::= "int" Ids ";" Stmt
13   syntax Ids ::= List{Id, ","}
14 endmodule
15
16 module IMP imports IMP-SYNTAX imports DOMAINS
17   syntax KResult ::= Int
18   configuration
19     <T> <k> $PGM:Pgm </k> <state> .Map </state> </T>
20   rule <k> X:Id => I ...</k> <state>... X |-> I ...</state>
21   rule I1 + I2 => I1 +Int I2
22   rule I1 - I2 => I1 -Int I2
23   rule <k> X = I; => . ...</k>
24        <state>... X |-> (J => I) ...</state>
25   rule S1:Stmt S2:Stmt => S1 ~> S2
26   rule if (I) S1 S2 => S1 requires I =/=Int 0
27   rule if (0) S1 S2 => S2
28   rule while(B) S => if(B) {S while(B) S} {}
29   rule {} => .
30   rule <k> int (X, Xs => Xs); S </k>
31        <state>... (. => X |-> 0) </state>
32   rule int .Ids; S => S
33 endmodule
```

Figure 2: The complete 𝕂 definition of IMP.

Certain frontend syntactic sugar is introduced to make 𝕂 rules more compact and modular, and to avoid duplication. For example, the variable lookup rule (line 20)

```
rule <k> X:Id => I ...</k> <state>... X |-> I ...</state>
```

puts the rewrite symbol => not at the top, but locally at where the rewrite happens, so we need not duplicate or even mention (thanks to the "..." notation) irrelevant parts of the configuration, called *structural frame*. Frontend sugar is important for 𝕂 to be scalable and applicable to defining formal semantics of large languages, but it is eliminated when parsing the definitions. Our interest here is the after-parsing 𝕂 features, that is, its logical foundation for defining transition systems as logical theories and capturing program execution and verification as rigorous logic reasoning.

1.2 Our Contribution: Applicative Matching Logic

The main technical contribution of this paper is the proposal of *applicative matching logic* (AML) as a new logic foundation of 𝕂. We show existing 𝕂 features (discussed in Section 1.1) can be defined as logical theories in AML. Specifically, we use AML to capture all the following logical frameworks

and systems that have been fully or partially supported by the current $\mathbb{K}$ implementations (also shown in Fig. 3):

- many-sorted first-order logic (MSFOL), constructors, term algebras, and the inductive principle (Section 4);
- MSFOL variants: algebraic specifications, FOL with partial functions, and FOL with fixpoints (Section 5);
- order-sorted algebras (OSA) (Section 6);
- parametric sorts and operations (Section 7).

Then we put everything together in Section 8 and show that program execution and verification in $\mathbb{K}$ can be regarded as best effort implementations of AML formal reasoning.

The development of $\mathbb{K}$ seems to be mostly driven by user requests. Many $\mathbb{K}$ features were engineered and optimized to ease the development of formal semantics of large languages (e.g., C, Java, JavaScript), rather than implemented following a well-established foundational theory. Even today, more than 15 years after its inception, $\mathbb{K}$ seems to be a moving target that is continuously evolving, according to the Github issues and feature requests [33, 34]. These make it difficult to find a mathematically solid foundation for $\mathbb{K}$. We hope this paper gives substantial evidence that AML can be taken as the foundation of $\mathbb{K}$, and due to its simplicity and flexibility, the $\mathbb{K}$ development team will benefit from doing so, because AML not only gives a rigorous mathematical explanation of all existing $\mathbb{K}$ features, but also establishes a solid foundation for developing new features. As an example, we propose two new features in Section 9, function sorts/types and dependent sorts/types, based on the foundation provided by AML.

The rest of the paper is organized as follows. We define the syntax and semantics of AML in Section 2 and its proof system in Section 3. Then we follow the above agenda: defining all logical systems in Fig. 3 in Sections 4-7, putting them together and formalizing $\mathbb{K}$ in Section 8, and proposing new features in Section 9. Finally, we discuss related work in Section 10 and conclude in Section 11.

Proofs for all the results are in the appendix

2 Applicative Matching Logic

Here we define the syntax and semantics of the novel AML.

2.1 Applicative Matching Logic Syntax

Definition 1. An AML *signature* $(\textsc{EVar}, \textsc{SVar}, \Sigma)$ has a set $\textsc{EVar}$ of *element variables* $x, y, \ldots$, a set $\textsc{SVar}$ of *set variables* $X, Y, \ldots$, and a set Σ of *(constant) symbols* $\sigma, f, g, \ldots$. We often omit $\textsc{EVar}$ and $\textsc{SVar}$ and use Σ to denote the signature. The set of AML *patterns*, denoted $\textsc{Pattern}$, is defined as:

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

where in $\mu X.\varphi$ we require that φ is positive in X, i.e., X is not nested in an odd number of times on the left of an implication $\varphi_1 \to \varphi_2$. Pattern $\varphi_1\varphi_2$ is called *application* and assumed associative to the

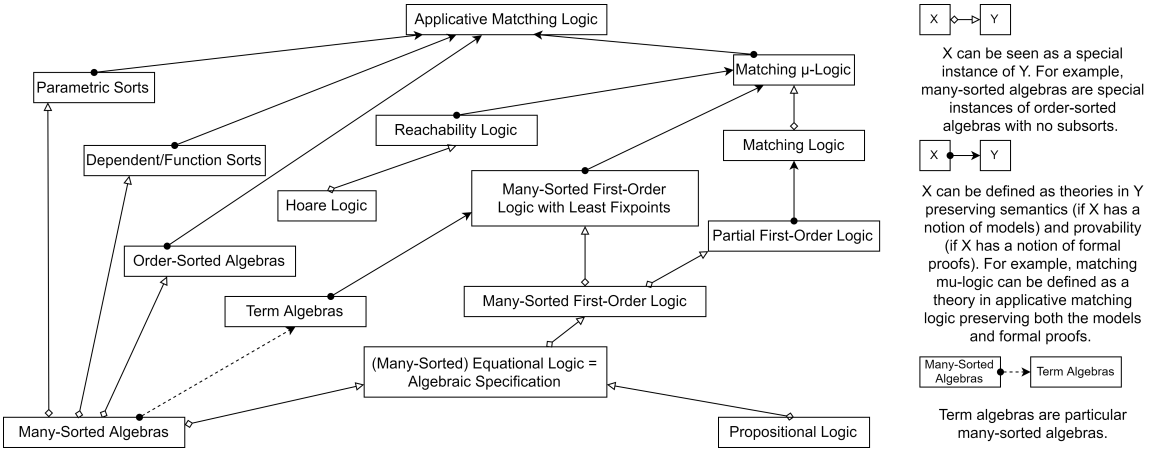

Figure 3: $\mathbb{K}$ implements, fully or partially, many logical frameworks that can be subsumed in AML as logical theories.

left. The scope of binders $\exists$ and μ goes farthest to the right. The notions of free variables $\mathrm{FV}(\varphi)$, α-renaming, and capture-avoiding substitution ($\varphi[\psi/x]$ and $\varphi[\psi/X]$) are defined in the usual way.

We argue that the above syntax is *minimal* while still being useful and expressive to formalize language semantics and program properties. Element variables are FOL-style variables that are necessary for ranging over individual elements, which can then be quantified (i.e., "abstracted") by the $\exists$ binder. Set variables are like propositional variables in modal logic that are necessary for ranging over sets (i.e., predicates), which can then be quantified by μ to create least fixpoints. Symbols are used to represent functions, predicates, constructors, and modal operators, in a uniform way. Together with application, symbols build complex patterns from simpler ones (i.e., $\sigma\varphi_1 \dots \varphi_n$), which can represent terms (e.g., $\sigma \equiv f$ for function f), FOL-style formulas (e.g., $\sigma \equiv p$ for predicate p), program configurations (e.g., σ being the <k/> cell), and modal formulas such as temporal and reachability properties (e.g., σ being the "next" operator $\bigcirc$ in LTL [51]). Note that application in AML is generic and can be instantiated with arbitrary patterns. This gives us maximal flexibility and expressiveness, and at the same time minimal syntax complexity, because we need not define multiary functions, predicates, constructors, or polyadic modal operators. Instead, all of them are uniformly treated using symbols and the generic application. Finally, we have the usual $\bot$ and $\varphi_1 \to \varphi_2$ to build arbitrary logical constraints.

We define the following derived constructs in the usual way. We assume the standard precedence among logical connectives, where application binds the tightest.

$$\neg\varphi \equiv \varphi \to \bot \qquad \varphi_1 \vee \varphi_2 \equiv \neg\varphi_1 \to \varphi_2 \qquad \varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$$
$$\top \equiv \neg\bot \qquad \forall x.\varphi \equiv \neg\exists x.\neg\varphi \qquad \varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$$
$$\nu X.\varphi \equiv \neg\mu X.\neg\varphi[\neg X/X] \quad /\!\!/ \text{ if } \varphi \text{ is positive in } X$$

2.2 Applicative Matching Logic Semantics

AML has a *pattern matching semantics*. Patterns are interpreted to the *sets* of elements that *match* them.

Let us first look at some examples. Pattern $\varphi_1 \wedge \varphi_2$ is matched by the elements that match both φ_1 and φ_2. Pattern $\neg\varphi$ is matched by the elements that do not match φ. Patterns $\top$ and $\bot$ are matched by all elements and no elements. Element variable x is matched by the unique element to which x evaluates (valuations are defined later). Set variable X is matched by the elements in the set to which X evaluates. Symbols also evaluate to sets, but their interpretations are given directly by AML structures, not valuations (structures are defined later).

The meaning of $\varphi_1\varphi_2$ depends on how we interpret φ_1 and φ_2. If φ_1 represents a *constructor* c and φ_2 represents its argument a, then $\varphi_1\varphi_2$ is matched by *term* $c(a)$. If φ_1 represent a *function* f and φ_2 represents its argument a, then $\varphi_1\varphi_2$ is matched by *value* $f(a)$. If φ_1 represents a *predicate* p and φ_2 represents its argument a, then $\varphi_1\varphi_2$ denotes *truth value* $p(a)$. If φ_1 represents a *dynamic relation*, e.g., the "next operator $\bigcirc$" in LTL [51], and φ_2 represents a state s, then $\varphi_1\varphi_2$ is matched by all states whose next states match φ_2. In practice, $\mathbb{K}$ uses constructors and terms to represent language concrete syntax and program configurations, uses functions to represent functions on mathematical domains, uses predicates to specify mathematical assertions, and uses dynamic relations to support program execution and verification. Thus, AML unifies constructors, functions, predicates, and dynamic operations used in $\mathbb{K}$, by application patterns.

The above interpretation can be generalized to the pattern matching semantics of AML. E.g., if φ_1 is matched by a *set* of constructors C and φ_2 is matched by a *set* of arguments A, then $\varphi_1\varphi_2$ is matched by all terms $c(a)$ with $c \in C$ and $a \in A$. Multiary constructors, functions, predicates, and modalities, can be uniformly defined by currying the application.

Binder $\exists$ builds *abstractions*. Pattern $\exists x.\varphi$ is matched by the elements matching φ for *some valuations* of x, i.e., it "abstracts away" the irrelevant part x from the matched part φ. Binder μ builds *least fixpoints*. Intuitively, a pattern $\varphi(X)$ defines a function $\mathcal{F}_{\varphi,X}$ over sets that maps X to $\varphi(X)$. The requirement that $\varphi(X)$ is positive in X implies that $\mathcal{F}_{\varphi,X}$ is monotone and has a unique least fixpoint $\mu\mathcal{F}_{\varphi,X}$, by the Knaster-Tarski fixpoint theorem (Theorem 19). Pattern $\mu X.\varphi$ is then matched by the elements in $\mu\mathcal{F}_{\varphi,X}$.

Definition 2. An AML structure $(M, \_\bullet\_, \{\sigma_M\}_{\sigma\in\Sigma})$ has:

- a nonempty set M, called *domain*;
- a binary *application* function $\_\bullet\_ \colon M \times M \to \mathcal{P}(M)$;
- a subset $\sigma_M \subseteq M$ called *interpretation* of $\sigma \in \Sigma$.

We use M to denote the above structure and also call it a *model*. We extend *pointwisely* the application function $\_\bullet\_$ from over elements to over sets, as follows (where $A, B \subseteq M$)

$$\_\bullet\_ : \mathcal{P}(M) \times \mathcal{P}(M) \to \mathcal{P}(M) \qquad\qquad A \bullet B = \textstyle\bigcup_{a\in A, b\in B} a \bullet b$$

Note that $A \bullet B = \emptyset$ whenever $A = \emptyset$ or $B = \emptyset$. We abbreviate $a \bullet b$ as ab and write $ab_1 \cdots b_n \equiv (\cdots((ab_1)\, b_2)\cdots b_n)$.

Definition 3. Given $(\textsc{EVar}, \textsc{SVar}, \Sigma)$ and a structure M, a *valuation* is a function $\rho \colon (\textsc{EVar} \cup \textsc{SVar}) \to (M \cup \mathcal{P}(M))$ with $\rho(x) \in M$ for all $x \in \textsc{EVar}$ and $\rho(X) \subseteq M$ for all $X \in \textsc{SVar}$. Its *extension*, $\bar{\rho} \colon \textsc{Pattern} \to \mathcal{P}(M)$, is defined as:

$\bar{\rho}(x) = \{\rho(x)\}$ for $x \in \text{EVAR}$
$\bar{\rho}(X) = \rho(X)$ for $X \in \text{SVAR}$
$\bar{\rho}(\bot) = \emptyset$
$\bar{\rho}(\sigma) = \sigma_M$ for $\sigma \in \Sigma$
$\bar{\rho}(\varphi_1\varphi_2) = \bar{\rho}(\varphi_1)\bar{\rho}(\varphi_2)$
$\bar{\rho}(\varphi_1 \to \varphi_2) = M \setminus (\bar{\rho}(\varphi_1) \setminus \bar{\rho}(\varphi_2))$
$\bar{\rho}(\exists x.\varphi) = \bigcup_{a \in M} \overline{\rho[a/x]}(\varphi)$
$\bar{\rho}(\mu X.\varphi) = \mu\mathcal{F}^{\rho}_{\varphi,X}$ where $\mathcal{F}^{\rho}_{\varphi,X}(A) = \overline{\rho[A/X]}(\varphi)$ for $A \subseteq M$
Here, "\" is set difference; $\rho[a/x]$ (resp. $\rho[A/X]$) denotes the valuation ρ' such that $\rho'(x) = a$ (resp. $\rho'(X) = A$) and agrees with ρ on all other variables. The existence of $\mu\mathcal{F}^{\rho}_{\varphi,X}$ is by the Knaster-Tarski fixpoint theorem (Theorem 19).

AML patterns are not two-valued. They can evaluate to $\emptyset$, or M (i.e., the entire domain), or any subsets in between. However, we can easily restore the classic two-valued semantics by letting $\emptyset$ denote "logical false" and M denote "logical true". Since M is nonempty, no confusion is possible. We call φ a *predicate*, if $\bar{\rho}(\varphi) \in \{\emptyset, M\}$ for all ρ.

Proposition 4. The derived constructs defined in Section 2.1 have the expected semantics:
$\bar{\rho}(\neg\varphi) = M \setminus \bar{\rho}(\varphi)$
$\bar{\rho}(\varphi_1 \vee \varphi_2) = \bar{\rho}(\varphi_1) \cup \bar{\rho}(\varphi_2)$
$\bar{\rho}(\top) = M$
$\bar{\rho}(\varphi_1 \wedge \varphi_2) = \bar{\rho}(\varphi_1) \cap \bar{\rho}(\varphi_2)$
$\bar{\rho}(\forall x.\varphi) = \bigcap_{a \in M} \overline{\rho[a/x]}(\varphi)$
$\bar{\rho}(\varphi_1 \leftrightarrow \varphi_2) = M \setminus (\bar{\rho}(\varphi_1) \triangle \bar{\rho}(\varphi_2))$
$\bar{\rho}(\nu X.\varphi) = \nu\mathcal{F}^{\rho}_{\varphi,X}$ with $\mathcal{F}^{\rho}_{\varphi,X}$ defined as in Definition 3
where set symmetric difference $A \triangle B = (A \setminus B) \cup (B \setminus A)$.

The "predicate-ness" of patterns is preserved by all the above constructs. E.g., $\varphi_1 \to \varphi_2$ and $\varphi_1 \wedge \varphi_2$ are predicates whenever φ_1 and φ_2 are, $\mu X.\varphi$ is a predicate whenever φ is; etc.

What does it mean for a model M to *satisfy* φ? In classic two-valued logics such as FOL, M satisfies φ, iff φ evaluates to "true" in M, under all valuations. In AML, the logical true is represented by evaluating to the entire domain M, so naturally we have the following definition of satisfaction.

Definition 5. We say M *satisfies* φ, written as $M \vDash \varphi$, if $\bar{\rho}(\varphi) = M$ for all ρ. Let Γ be a set of patterns. We write $M \vDash \Gamma$ iff $M \vDash \psi$ for all $\psi \in \Gamma$, and $\Gamma \vDash \varphi$ iff $M \vDash \Gamma$ implies $M \vDash \varphi$ for all M. Using conventional terminology, we call Γ a *theory*, its elements *axioms*, and M a *model* of Γ whenever $M \vDash \Gamma$.

Axioms and theories are used to *restrict* the models, so instead of considering all AML structures, we only consider those satisfy the axioms. In the following, when we say that we define/assume/introduce a symbol σ and/or an axiom φ, we mean to add σ to Σ and/or add φ to Γ, and consider models of the thus extended Γ. In Section 3, we will introduce the proof system and proof theory of AML. In there, axioms and theories are also used for *formal reasoning*.

2.3 Important Mathematical Instruments

Here we show how to define several important mathematical instruments, such as equality, membership, functions, and tuples, as notations and theories in AML.

Definition 6. Let $\lceil \_ \rceil$ be a symbol, which we call the *definedness* symbol. We write $\lceil \varphi \rceil$ instead of $\lceil \_ \rceil\, \varphi$. Also, let (Definedness) be the axiom $\forall x.\lceil x \rceil$, or just $\lceil x \rceil$. We introduce the following important notations:

$$\begin{array}{ll} \text{totality } \lfloor \varphi \rfloor \equiv \neg\lceil\neg\varphi\rceil & \text{equality } \varphi_1 = \varphi_2 \equiv \lfloor \varphi_1 \leftrightarrow \varphi_2 \rfloor \\ \text{membership } x \in \varphi \equiv \lceil x \wedge \varphi \rceil & \text{inclusion } \varphi_1 \subseteq \varphi_2 \equiv \lfloor \varphi_1 \rightarrow \varphi_2 \rfloor \end{array}$$

$$\varphi_1 \neq \varphi_2 \equiv \neg(\varphi_1 = \varphi_2) \quad x \notin \varphi \equiv \neg(x \in \varphi) \quad \varphi_1 \not\subseteq \varphi_2 \equiv \neg(\varphi_1 \subseteq \varphi_2)$$

For convenience, we tacitly assume the definedness symbol and the (Definedness) axiom, as well as all above notations. Note that we are *not* extending AML, but simply adding one symbol and one axiom to all subsequent AML theories defined in this paper. In other words, we only consider models that satisfy (Definedness), for example, a model M with a special element \$ such that $\$ \cdot a = M$ for all $a \in M$. By pointwise extension, $\$ \cdot A = M$ iff $A \neq \emptyset$, so we can use \$ and application to check if A is nonempty. In other words, $\lceil \varphi \rceil$ is a predicate stating that φ is *defined*, i.e., φ is matched by some elements.

It is straightforward to show all notations in Definition 6 are predicates with the expected semantics (see Proposition 20). E.g., $\bar{\rho}(\lfloor \varphi \rfloor) = M$ iff $\bar{\rho}(\neg\lceil\neg\varphi\rceil) = M$, iff $\bar{\rho}(\lceil\neg\varphi\rceil) = \emptyset$, iff $\bar{\rho}(\neg\varphi) = \emptyset$, iff $\bar{\rho}(\varphi) = M$, i.e., φ is total. Another example: $\bar{\rho}(\varphi_1 = \varphi_2) = M$ iff $\bar{\rho}(\lfloor \varphi_1 \leftrightarrow \varphi_2 \rfloor) = M$, iff $\bar{\rho}(\varphi_1 \leftrightarrow \varphi_2) = M$, iff $\bar{\rho}(\varphi_1) \mathbin{\triangle} \bar{\rho}(\varphi_2) = \emptyset$, iff $\bar{\rho}(\varphi_1) = \bar{\rho}(\varphi_2)$, i.e., φ_1 equals φ_2; etc.

Definedness and the related notations are important in writing axioms and defining theories. For example, the following axiom defines *functional constant symbols*, which are symbols $\sigma \in \Sigma$ that are forced to be interpreted to *singleton sets* (instead of any subsets) in all models satisfying it

$$\text{(Functional Constant)} \quad \exists z.\, \sigma = z$$

This is because, intuitively, that z is an element variable whose semantics is always a singleton set. By $\sigma = z$, we let σ equal the singleton set denoted by z, and then abstract away z by $\exists z$, so σ is any singleton set (but not any subset).

3 AML Proof System

We present the AML proof system, inspired by [9]. We first define *application contexts*, written C, as patterns with a distinguished placeholder variable $\Box$ such that the rooted path to $\Box$ has only applications. We abbreviate $C[\varphi/\Box]$ as $C[\varphi]$, to denote the result of plugging φ into C.

The Hilbert-style proof system of AML is shown in Fig. 4. It consists of four categories of proof rules. The first contains four rules that provide *complete FOL reasoning* [58], which justifies the use of SAT/SMT solvers [17, 2] in $\mathbb{K}$. The second contains four rules that provide *frame reasoning* over application contexts. Since $\mathbb{K}$'s configurations and cells are built with symbols and the generic application, this set of proof rules supports *compositional reasoning* in $\mathbb{K}$. The third contains three rules for *fixpoint reasoning* [36], where (Knaster-Tarski), also known as Park induction [19], is a logical incarnation of the Knaster-Tarski fixpoint theorem (Theorem 19) into AML and is the key to reasoning about inductively-defined data structures and dynamic properties such as program reachability. Finally, we need two technical rules for certain completeness results [9]. The provability relation is denoted $\Gamma \vdash \varphi$, which means that φ can be proved by the proof system with patterns in Γ as additional axioms.

Equational reasoning is sound in AML for theories with definedness, from which equality is derived (see Definition 6).

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

Figure 4: The Hilbert-style proof system of applicative matching logic (where C, C_1 and C_2 are application contexts as defined at the beginning of this section.

Proposition 7. For theories Γ with definedness, we have

- $\Gamma \vdash \varphi = \varphi$
- $\Gamma \vdash \varphi_1 = \varphi_2$ and $\Gamma \vdash \varphi_2 = \varphi_3$ implies $\Gamma \vdash \varphi_1 = \varphi_3$
- $\Gamma \vdash \varphi_1 = \varphi_2$ implies $\Gamma \vdash \varphi_2 = \varphi_1$
- $\Gamma \vdash \varphi_1 = \varphi_2$ implies $\Gamma \vdash \psi[\varphi_1/x] = \psi[\varphi_2/x]$

Here ψ can be any pattern, not only application context. In other words, the *indiscernibility of identicals* holds in AML.

Hence, AML supports FOL reasoning, frame reasoning, fixpoint reasoning, and equational reasoning. This forms the foundation of formal reasoning and analysis in $\mathbb{K}$.

Theorem 8 (Soundness Theorem)**.** $\Gamma \vdash \varphi$ *implies* $\Gamma \models \varphi$.

However, AML does not have any complete proof system. In Section 4.5, we define a finite AML theory that captures $(\mathbb{N}, +, \times)$, the standard model of natural numbers with addition and multiplication, *up to isomorphism*. By Gödel's first incompleteness theorem [22], any effective and

sound proof system of AML cannot be complete for all theories. In practice, the proof system in Fig. 4 is sufficient to do formal proofs supported by the logical systems in Fig. 3 and to formalize program execution and verification in $\mathbb{K}$ (see Section 8).

⋆ Agenda for the Rest of the Paper

We remind the readers that our main objective, as stated in Section 1.2, is to define all the existing features of $\mathbb{K}$ and the logical systems implemented by $\mathbb{K}$, as theories and notations in AML. The challenge is essentially to show how an inherently unsorted logic like AML can deal with *sorts* in their full generality, including constructors and inductively defined data-types, partiality and subsorting, and parametric sorts and operations. We discuss these in Sections 4-7. In Section 8, we put all these together in the context of the semantics of the existing $\mathbb{K}$, and in Section 9, backed by AML semantics, we propose two new features of $\mathbb{K}$ requested by users.

4 Instance: Many-Sorted First-Order Logic

We use *many-sorted first-order logic* (abbreviated MSFOL) as an example to discuss in detail how to deal with sorts in the unsorted AML. We consider both model-theoretic and proof-theoretic aspects. Model-theoretically, we capture MSFOL *structures*, which are many-sorted algebras (abbreviated MSA) accompanied by multiary relations as interpretations of MSFOL predicates. Proof-theoretically, we show how to re-produce MSFOL formal proofs within the AML proof system. The main technical contribution is the conservative extension theorem, which states that the semantics and provability of MSFOL and the corresponding AML theory, are all equivalent (see Theorem 16).

This is a core section. The techniques used here will be repeatedly used later. Our major objective is to demonstrate that AML, in spite of its simplicity, can recover, as notations and theories, other important, familiar, but notationally heavier concepts. MSFOL has built-in systematic mechanisms and mathematical instruments for sorts and many-sorted structures. These make MSFOL complex, and also too *rigid* to adapt to other variants or extension, e.g., order-sorted functions, parametric operations, and partial functions. In contrast, AML takes a lighter approach by requiring only the minimal, yet powerful logical infrastructure, this way giving maximal flexibility in defining the necessary mathematical instruments, e.g., those for sorts and many-sorted functions, as theories and notations with patterns as axioms, which are intuitive, simple, highly adaptive, and easily extensible.

4.1 MSFOL Preliminaries

Definition 9 ([42, 14])**.** An MSFOL signature (S, F, Π) has a sort set S, an $(S^* \times S)$-indexed set $F = \{F_{s_1 \ldots s_n, s}\}_{s_1, \ldots, s_n, s \in S}$ of functions, and an S^*-indexed set $\Pi = \{\Pi_{s_1 \ldots s_n}\}_{s_1, \ldots, s_n \in S}$ of predicates. We use $x{:}s$ to denote sorted variables. Sorted terms and formulas are inductively defined as follows:

$$t_s ::= x{:}s \mid f(t_{s_1}, \ldots, t_{s_n}) \text{ for } f \in F_{s_1 \ldots s_n, s}$$
$$\varphi ::= p(t_{s_1}, \ldots, t_{s_n}) \text{ for } p \in \Pi_{s_1 \ldots s_n} \mid \bot \mid \varphi_1 \to \varphi_2 \mid \exists x{:}s.\, \varphi$$

An MSFOL structure $A = (\{A_s\}_{s \in S}, \{f_A\}_{f \in \Sigma}, \{p_A\}_{p \in \Pi})$ has a nonempty domain A_s for each $s \in S$, an interpretation $f_A : A_{s_1} \times \cdots \times A_{s_n} \to A_s$ for each $f \in F_{s_1 \ldots s_n, s}$, and an interpretation $p_A \subseteq A_{s_1} \times \cdots \times A_{s_n}$ for each $p \in \Pi_{s_1 \ldots s_n}$. Valuations of variables (denoted v) and interpretations of terms and formulas

(denoted $\bar{v}(t_s) \in A_s$ and $\bar{v}(\varphi) \in \{true, false\}$, resp.) are defined as usual; see [42] for details. We write $A \vDash_{\mathsf{MSFOL}} \varphi$ iff $\bar{v}(\varphi) = true$ for all v. An MSFOL theory Ω is a set of MSFOL formulas. We define $\Omega \vDash_{\mathsf{MSFOL}} \varphi$ iff $A \vDash_{\mathsf{MSFOL}} \Omega$ implies $A \vDash_{\mathsf{MSFOL}} \varphi$ for all A. We define $\Omega \vdash_{\mathsf{MSFOL}} \varphi$ iff φ can be proved with formulas in Ω as additional axioms, by the MSFOL proof system, which has the same FOL reasoning rules as AML (Fig. 4) except (∃-Quantifier) and (∃-Generalization), which are changed to the sorted versions (assuming $x{:}s \notin \mathrm{FV}(\varphi_2)$):

$$\text{(Sorted ∃-Quantifier)}\ \ \varphi[t_s/x{:}s] \to \exists x{:}s.\,\varphi \qquad \text{(Sorted ∃-Generalization)}\ \ \frac{\varphi_1 \to \varphi_2}{(\exists x{:}s.\,\varphi_1) \to \varphi_2}$$

4.2 Defining MSFOL Syntax and Proofs in AML

Fix an MSFOL signature (S, F, Π) and an MSFOL theory Ω. We define an AML signature Σ^{MSFOL} and an AML theory Γ^{MSFOL} that precisely captures Ω, both proof-theoretically and model-theoretically. We follow the classic *sort-as-predicate* paradigm (see, e.g., [13, Section 5]), but thanks to the pattern matching semantics of AML, our definition is more succinct.

Sorts. We define a symbol $Sort \in \Sigma^{\mathsf{MSFOL}}$ to represent the sort set. For each $s \in S$, we define a corresponding functional constant $s \in \Sigma^{\mathsf{MSFOL}}$ (see Section 2.3), called a *sort constant* or a *sort*, and add the following axiom to Γ^{MSFOL} (for $s \in S$):

$$\text{(Sort)}\ \ s \in Sort\ \ /\!\!/ \text{ membership } \_\in\_ \text{ defined in Section 2.3}$$

We define a symbol $[\![\_]\!] \in \Sigma^{\mathsf{MSFOL}}$ called the *domain symbol* and use $[\![\_]\!]\, s$, written $[\![s]\!]$, to represent the domain of s. Intuitively, $[\![s]\!]$ is matched by all elements of sort s. Properties about sorts can be expressed by AML patterns. For example, $x \in [\![s]\!]$ states that x has sort s; $\varphi \subseteq [\![s]\!]$ states that all elements matching φ have sort s. The nonemptiness of the domain of s is axiomatized by ($\_\neq\_$ defined in Section 2.3):

$$\text{(Nonempty Domain)}\ \ [\![s]\!] \neq \bot \ \text{ for } s \in S$$

Sorted quantification. AML is unsorted. Its quantification $\exists x.\,\varphi$ ranges over all elements. MSFOL has sorted quantification $\exists x{:}s.\,\varphi$, which only ranges over elements of s. Therefore, we define sorted quantification as syntactic sugar in AML:

$$\exists x{:}s.\,\varphi \equiv \exists x.\,(x \in [\![s]\!]) \wedge \varphi \qquad\qquad \forall x{:}s.\,\varphi \equiv \forall x.\,(x \in [\![s]\!]) \to \varphi$$

Intuitively, $\exists x{:}s.\,\varphi$ is matched by φ for some x of sort s. The following shows the duality between $\exists x{:}s$ and $\forall x{:}s$ in AML.

Proposition 10. $\vdash \forall x{:}s.\,\varphi = \neg\exists x{:}s.\,\neg\varphi$

Many-sorted functions and terms. We translate MSFOL terms to AML patterns. For MSFOL variables $x{:}s$, we assume a corresponding AML variable denoted x and use the "well-sorted" predicate to specify its sort; see Definition 11. For MSFOL functions $f \in F_{s_1 \ldots s_n, s}$, we define a corresponding symbol $f \in \Sigma^{\mathsf{MSFOL}}$ and the next axiom to specify its arity:

$$\text{(Function)}\quad \forall x_1{:}s_1 \ldots \forall x_n{:}s_n.\,\exists y{:}s.\,f\ x_1\ \cdots\ x_n = y$$

We use the *functional notation* $f\colon s_1 \times \cdots \times s_n \to s$ as a shortcut for (Function) axiom. MSFOL sorted terms $f(t_{s_1}, \ldots, t_{s_n})$ are then defined in a "curried" way, as patterns $f\ t_{s_1}\ \ldots\ t_{s_n}$.

Many-sorted predicates and formulas. We translate MSFOL formulas to AML patterns. Logical connectives $\bot$ and $\to$ are translated to themselves. Quantification $\exists x{:}s\,.\,\varphi$ is translated to the sorted quantification in AML. For many-sorted predicates $p \in \Pi_{s_1 \dots s_n}$, we define a symbol p and the next axiom to specify that p is a predicate (see after Definition 3):

(Predicate)
$$\forall x_1{:}s_1 \dots \forall x_n{:}s_n\,.\,(p\,x_1\,\cdots\,x_n = \top) \lor (p\,x_1\,\cdots\,x_n = \bot)$$

Note the disjunction above cannot be replaced with $p\,x_1\,\cdots\,x_n \lor \neg(p\,x_1\,\cdots\,x_n)$, because the latter is vacuously $\top$ in AML. We use $p\,t_{s_1}\,\cdots\,t_{s_n}$ in AML to define MSFOL's $p(t_{s_1}, \dots, t_{s_n})$.

Let Γ^{MSFOL} contain all axioms so far. We need to show that it correctly captures the syntax of MSFOL terms and formulas. The arities of functions and predicates are axiomatized in Γ^{MSFOL} as shown above. The sorts of variables, however, were all dropped during the translation. We need to restore them, so we define the following "well-sorted" predicates.

Definition 11. For MSFOL terms t and formulas φ, let

$$\mathrm{ws}(t), \mathrm{ws}(\varphi) \equiv x_1 \in [\![s_1]\!] \land \cdots \land x_n \in [\![s_n]\!]$$

be the well-sorted predicates, where $x_1{:}s_1, \dots, x_n{:}s_n$ are all free variables appearing in t or φ, respectively.

The following shows that MSFOL terms/formulas are wellformed patterns/predicates, if variables have intended sorts.

Proposition 12. For MSFOL terms t_s and formulas φ,

$$\Gamma^{\mathsf{MSFOL}} \vdash \mathrm{ws}(t_s) \to \exists y{:}s\,.\,t_s = y$$
$$\Gamma^{\mathsf{MSFOL}} \vdash \mathrm{ws}(\varphi) \to (\varphi = \top) \lor (\varphi = \bot).$$

MSFOL theories. Finally, we translate MSFOL theories to AML theories, using Proposition 12. Given an MSFOL axiom φ, we define its AML translation $\varphi^{\mathsf{MSFOL}} \equiv \mathrm{ws}(\varphi) \to (\varphi = \top)$. Given MSFOL theory Ω, we define its AML translation $\Omega^{\mathsf{MSFOL}} = \{\psi^{\mathsf{MSFOL}} \mid \psi \in \Omega\}$. Let Γ^{MSFOL} include all the translated axioms Ω^{MSFOL} as well. The following shows that Γ^{MSFOL} preserves MSFOL reasoning within Ω.

Theorem 13. $\Omega \vdash_{\mathsf{MSFOL}} \varphi$ *implies* $\Gamma^{\mathsf{MSFOL}} \vdash \varphi^{\mathsf{MSFOL}}$.

4.3 Subsuming MSFOL Models in AML

So far we have shown that the AML axiomatization Γ^{MSFOL} captures the MSFOL theory Ω in the sense that it preserves its provability. Next, we show that the MSFOL models of Ω are subsumed by the AML models of Γ^{MSFOL}, which leads us to the conservative extension theorem (Theorem 16).

Since AML models make no distinction among elements, sorts, functions, or predicates, they contain both the elements in MSFOL models and their sorts, functions, and predicates. To compare with MSFOL models, we need to *restrict* the AML models, as formalized below.

Definition 14. Let (S, F, Π) be an MSFOL signature and $(M, \_\bullet\_, \{\sigma_M\}_{\sigma \in \Sigma^{\mathsf{MSFOL}}})$ be an AML model of Γ^{MSFOL}. Its *MSFOL-restricted model* $A = (\{A_s\}_{s \in S}, \{f_A\}_{f \in F}, \{p_A\}_{p \in \Pi})$ is an MSFOL structure with

- a domain $A_s = [\![s_M]\!]_M$ for $s \in S$, where s_M is the interpretation of $s \in \Sigma^{\mathsf{MSFOL}}$ in M and $[\![s_M]\!]_M \equiv [\![\_]\!]_M \bullet s_M$;
- an interpretation $f_A : A_{s_1} \times \cdots \times A_{s_n} \to A_s$ for $f \in F_{s_1 \ldots s_n, s}$, defined by $\{f_A(a_1, \ldots, a_n)\} = f_M \bullet a_1 \bullet \cdots \bullet a_n$ for $a_1 \in A_{s_1}, \ldots, a_n \in A_{s_n}$, where f_M is the interpretation of $f \in \Sigma^{\mathsf{MSFOL}}$ in M;
- an interpretation $p_A = \{(a_1, \ldots, a_n) \mid p_M \bullet a_1 \bullet \cdots \bullet a_n = M, a_1 \in A_{s_1}, \ldots, a_n \in A_{s_n}\}$ for $p \in \Pi_{s_1 \ldots s_n}$.

The MSFOL structure A is well-defined. Its domain A_s is nonempty, due to axiom (Nonempty Domain). Its interpretations f_A are total functions, due to axiom (Function) that forces f_M to return singletons on all elements of the appropriate sorts. The following theorem shows that Γ^{MSFOL} captures precisely the MSFOL models of Ω when considering its MSFOL-restricted models.

Theorem 15. *For MSFOL theory Ω and model $A \models_{\mathsf{MSFOL}} \Omega$, there is an AML model $M \models \Gamma^{\mathsf{MSFOL}}$ whose MSFOL-restricted model is exactly A. In addition, $\Gamma^{\mathsf{MSFOL}} \vDash \varphi^{\mathsf{MSFOL}}$ iff $\Omega \vDash_{\mathsf{MSFOL}} \varphi$.*

Theorems 13 and 15 together yield the following conservative extension theorem for MSFOL.

Theorem 16. *For any MSFOL theory Ω and formula φ, the next diagram holds (where "$X \Longrightarrow Y$" means "X implies Y"):*

$$\begin{array}{ccc} \Omega \vdash_{\mathsf{MSFOL}} \varphi & \Longrightarrow_1 & \Gamma^{\mathsf{MSFOL}} \vdash \varphi \\ \Uparrow_4 & & \Downarrow_2 \\ \Omega \vDash_{\mathsf{MSFOL}} \varphi & \Longleftarrow_3 & \Gamma^{\mathsf{MSFOL}} \vDash \varphi \end{array}$$

Proof. ($\Longrightarrow_1$) is by Theorem 13. ($\Longrightarrow_2$) is by Theorem 8 (i.e., the soundness of AML). ($\Longrightarrow_3$) is by Theorem 15, where we only need the direction from AML to MSFOL. ($\Longrightarrow_4$) is by the completeness of MSFOL (see, e.g., [42, 14]). □

4.4 Example: Natural Numbers (without Induction)

To emphasize that in spite of its generality AML recovers the familiar MSFOL-specific notions and notations, here we show an example of defining an AML theory Γ^{NAT0} that defines $(\mathbb{N}, +, \times)$, i.e. natural numbers with addition and multiplication. We do not yet consider the inductive principle.

Let *Sort* be a symbol that represents the sort set and *Nat* be a sort symbol, axiomatized by $Nat \in Sort$, like in Section 4.2. We define the following self-explanatory functions about natural numbers (function notations are defined in Section 4.2):

$$\begin{array}{ll} 0 : \to Nat & succ : Nat \to Nat \\ plus : Nat \times Nat \to Nat & mult : Nat \times Nat \to Nat \end{array}$$

where $0 : \to Nat$ means 0 is a nullary function, i.e., a functional constant of *Nat*. The above functions are axiomatized by the expected patterns/axioms, e.g.:

$$\forall x : Nat \,.\, plus\ x\ 0 = x.$$

For brevity, we omit the remaining axioms and call the resulting theory Γ^{NAT0}. Of course, Γ^{NAT0} is not a complete axiomatization, because it does not support *inductive reasoning*.

4.5 Example: Natural Numbers (with Induction)

Here we show that by adding three intuitive axioms to Γ^{NAT0} we obtain a (finite) AML axiomatization of natural numbers that allows us to derive the (infinite) Peano inductive theory as theorems. We extend Γ^{NAT0} by the *"no confusion" axioms* for 0 and *succ*, and most importantly, the *inductive principle*:

$$\begin{aligned}
&(\textsc{No Confusion}) \quad \forall x{:}\mathit{Nat}\,.\,\mathit{succ}\,x \neq 0\\
&(\textsc{No Confusion}) \quad \forall x{:}\mathit{Nat}\,.\,\forall y{:}\mathit{Nat}\,.\,\mathit{succ}\,x = \mathit{succ}\,y \to x = y\\
&(\textsc{Inductive Domain}) \quad [\![\mathit{Nat}]\!] = \mu D\,.\,0 \vee \mathit{succ}\,D
\end{aligned}$$

We call the resulting theory Γ^{NAT}. Intuitively, (No Confusion I) says that 0 and *succ* build different terms; (No Confusion II) says that *succ* is an injective function; (Inductive Domain) forces $[\![\mathit{Nat}]\!]$ to be the smallest set closed under 0 and *succ*, yielding exactly the standard natural numbers $\mathbb{N}$.

Peano induction [43, 50] in Peano arithmetic, shown below, can be proved from (Inductive Domain) as AML theorems.

$$\begin{aligned}
&(\textsc{Peano Induction})\\
&\varphi(0) \wedge (\forall y\,.\,\varphi(y) \to \varphi(\mathit{succ}(y))) \to \forall x\,.\,\varphi(x)
\end{aligned}$$

Recall that Peano arithmetic is a FOL theory of $(\mathbb{N}, +, \times)$, whose formulas are built from equations and FOL connectives. Since both are already defined in AML notationally, Peano arithmetic formulas are well-formed AML patterns. Peano induction is an *axiom schema* defined for each FOL formula $\varphi(x)$, while in AML, we can achieve its effect with only one pattern theorem (not a schema):

Proposition 17. $\Gamma^{\mathsf{NAT}} \vdash 0 \in X \wedge (\forall y{:}\mathit{Nat}\,.\,y \in X \to (\mathit{succ}\,y) \in X) \to \forall x{:}\mathit{Nat}\,.\,x \in X$.

Intuitively, it says if X is closed under 0 and *succ*, then it contains all natural numbers. All instances of (Peano Induction) can then be proved from this pattern theorem. The proof idea is to let $\Psi \equiv \exists z{:}\mathit{Nat}\,.\,z \wedge \varphi(z)$ be the pattern of all z such that $\varphi(z)$ holds. Then by standard AML reasoning, $\Gamma^{\mathsf{NAT}} \vdash (x \in \Psi) = \varphi(x)$, which reduces (Peano Induction) to Proposition 17, using rule (Set Variable Substitution) in Fig. 4, where ψ is substituted for X.

Finally, we show that Γ^{NAT} captures precisely $(\mathbb{N}, +, \times)$.

Proposition 18. The MSFOL-restricted model M of Γ^{NAT} is exactly $(\mathbb{N}, +, \times)$, up to isomorphism.

The main purpose of this section was to show that it is easy to define sorts and the related concepts such as many-sorted functions and predicates, in the unsorted AML. Not only can we capture precisely MSFOL (Theorem 16), both proof- and model-theoretically, but we can also define inductively-defined structures such as $(\mathbb{N}, +, \times)$. The latter is an example of *term algebras*, which are many-sorted structures whose domains are the smallest set that is inductively generated from a set of functions, or *constructors*. Term algebras represent an important instance of *initial algebra semantics* [24, 27], which has important applications in the foundations of programming language semantics and leads to various practical tools [27, 28, 12]). The technique that we used above to define $(\mathbb{N}, +, \times)$ can be easily generalized to term algebras; see Appendix F for details.

5 Instance: MSFOL Variants

Here, we discuss MSFOL variants that play an important role in programming language semantics, and thus $\mathbb{K}$, and not only, to show the flexibility and extensibility of AML. Indeed, we can re-use most of the theory Γ^{MSFOL} for MSFOL (shown in Section 4), with the minimal, necessary changes.

Algebraic specifications. These are commonly used to formalize abstract data types [40, 24] such as lists, sets, maps, graphs, stacks, queues, etc., and have led to various implementations in algebraic specification languages [12, 46, 18, 8]. An algebraic specification has a MSFOL signature $(S, F, \{=\})$ where equality is the only predicate, plus a set of equations E over MSFOL terms. Both sides of an equation have the same sort, and variables are assumed universally quantified over the entire equation. The models, called *F-algebras*, are MSFOL models where "=" is interpreted as the identity relation. We write $E \vDash_{\mathsf{Alg}} e$ iff any F-algebra A satisfying all equations in E also satisfies e.

Algebraic specifications are special instances of MSFOL theories, whose only predicate is equality and all axioms are equations. By Theorem 16, algebraic specifications are precisely captured in AML, by letting all functions in F be many-sorted functions with their (Function) axioms (see Section 4.2), and all equations in E be equational patterns/axioms (see Definition 6). The resulting AML theory, written E^{Alg}, then precisely captures algebraic specifications: $E^{\mathsf{Alg}} \vDash e$ iff $E \vDash_{\mathsf{Alg}} e$.

Partial functions. In MSFOL, function symbols are interpreted as total functions, which are common but can cause inconvenience when defining formal semantics; e.g., what is *head*(*list*) interpreted to, when *list* is empty? Partial MSFOL [20] solves this inconvenience by considering *partial functions* that can be *undefined* on some arguments. Partial MSFOL needs a different formalization than MSFOL to properly capture the desired properties of *definedness* [20].

On the other hand, the pattern matching semantics of AML easily captures definedness as the ability to match at least one element (recall Definition 6). Indeed, we can axiomatize a partial function $f : s_1 \times \cdots \times s_n \rightharpoonup s$ by adjusting the (Function) axiom for total functions (Section 4.2), to

$$\text{(Partial Function)} \quad \forall x_1{:}s_1 \ldots \forall x_n{:}s_n . \exists y{:}s . f\ x_1\ \cdots\ x_n \subseteq y$$

where the only change is from equality $=$ to inclusion $\subseteq$. Intuitively, the axiom enforces $f\ x_1\ \cdots\ x_n$ to return either a singleton (y), or nothing (the empty set), the latter meaning that f is undefined on $x_1, \ldots, x_n$.

Sorted fixpoints. Many MSFOL variants add support for *recursion* or *fixpoints* that are sorted. For example, LFP [29] extends MSFOL with *many-sorted recursive predicates*

$$p(x_1{:}s_1, \ldots, x_n{:}s_n) =_{lfp} \Phi(x_1{:}s_1, \ldots, x_n{:}s_n),$$

where p may occur positively, recursively in Φ, and "$=_{lfp}$" means that p is interpreted as the *least predicate*[1] satisfying the equation. Sorted recursion and fixpoints play an important role in defining inductive and coinductive data structures as well as program dynamic behaviors.

Matching (μ-)logic [9] (MmL) is an important MSFOL variant that supports *sorted fixpoints* $\mu X{:}s . \varphi$. MmL represents one of the latest attempts in formalizing $\mathbb{K}$ (see Section 10) and it captures many logical systems with recursion or fixpoints, including LFP. Therefore, we use MmL

[1] Where p is "less than" q iff q holds whenever p holds.

as an example to show how to define sorted fixpoints $\mu X{:}s.\varphi$ in AML. Interestingly, and perhaps unexpectedly, we need not do anything, besides using the "well-sorted" patterns (see Definition 11) to constrain the sorted variables in φ. We show that $\mu X.\varphi$ is neither bigger nor smaller than $\mu X{:}s.\varphi$. Firstly, $\mu X.\varphi$ denotes the least solution of $X = \varphi$ among *all subsets*, while $\mu X{:}s.\varphi$ denotes the least solution among *subsets of sort* s, so $\mu X.\varphi$ is no bigger than $\mu X{:}s.\varphi$. This also tells us that $\mu X.\varphi$ is of sort s, too, so it is no smaller than $\mu X{:}s.\varphi$ either, which is the *smallest* solution of sort s. Therefore, in the presence of the wellsortedness constraints for φ, which are always required, the AML unsorted fixpoint $\mu X.\varphi$ captures the same semantics as the MmL sorted fixpoint $\mu X{:}s.\varphi$.

Due to the importance of MmL, we proved that the entire MmL, including syntax, semantics, and formal proofs, can be captured in AML. As a result, all logical frameworks and systems subsumed by MmL (see Fig. 3) are also subsumed by AML. We include the (difficult) proof details in Appendix G.

6 Instance: Order-Sorted Algebras

𝕂 has built-in support for subsorting and symbol overloading. Even IMP, which is a trivial language, has three subsorts (Fig.2, lines 2 and 17). Real languages tend to have dozens of subsorts and overloaded symbols, such as various types of expressions and values with lists over them. According to the GitHub issues and discussions [33, 34], dealing with subsorting and symbol overloading *correctly and efficiently* is one of the most challenging problems that the development of 𝕂 faced. Using coercion functions (described at the end of this section) yields a correct but inefficient implementation, which would make 𝕂 useless with real languages like C, Java or JavaScript. Using specialized order-sorted algebra decision procedures for unification and rewriting, like OBJ [23] and Maude [12], yields an efficient but too hermetic to extend and too complex to trust implementation. Instead, the 𝕂 team has decided to follow an ad hoc solution based on a series of heuristics that work efficiently only with common cases, but whose correctness can be easily explained with the two types of axioms discussed below. New heuristics are added continuously, as users of 𝕂 discover new situations of interest. The incapacity of matching μ-logic [9] to deal with subsorting, and thus to serve as a logical foundation of 𝕂, was one of the main triggers for the development of AML. Below we show how naturally AML can handle subsorting by means of showing how it captures order-sorted algebra.

Order-sorted algebras (OSA) extend many-sorted algebras with a partial ordering $\leq \subseteq S \times S$ on sorts, called *subsort relation*, which forces a *subset relation* on the domain sets: $A_s \subseteq A_{s'}$ if $s \leq s'$. There are many OSA presentations, e.g., Goguen and Meseguer's [26, 44], Bidoit and Hennicker's [1, 31], and Axel Poigné's [52], which yield dozens of different OSA definitions (see, e.g., [25] for a survey) with slightly different requirements on the formalization. For concreteness, we consider the variant in [44] that allows *subsort overloading*, i.e., $f^{s_1 \ldots s_n, s} \in F_{s_1 \ldots s_n, s}$ and $f^{s'_1 \ldots s'_n, s'} \in F_{s'_1 \ldots s'_n, s'}$ with the same f and n but different argument and result sorts.

To define OSA as an AML theory, we reuse the theory of MSFOL (Section 4) and extend it to handle subsorting and symbol overloading. For the former, we define:

$$\text{(Subsort)} \quad [\![s]\!] \subseteq [\![s']\!] \quad \text{for all } s, s' \in S \text{ with } s \leq s'$$

to specify that the domain of s is included in that of s'. For the latter, we define symbol set $\{f \mid f^{s_1 \ldots s_n, s} \in F\}$ where we use one f for all its overloaded copies of various arity annotations. Then, we axiomatize that f is a function of the appropriate arities as in Section 4, but now f may have

multiple axioms, one for each overloaded copy $f^{s_1 \dots s_n, s} \in F_{s_1 \dots s_n, s}$:

$$\text{(Function)} \quad \forall x_1{:}s_1 \dots \forall x_n{:}s_n \,.\, \exists y{:}s \,.\, f x_1 \dots x_n = y$$

With the above definitions, we proved AML captures precisely OSA (see Theorem 32). We do not consider formal proofs in OSA, because unlike MSFOL, it does not appear to have a broadly-accepted notion of Hilbert-style proof system in literature. However, we point out that our definition of OSA naturally yields a complete proof system for OSA, *for free*, because one can use the AML proof system and the axioms for OSA, to formally reason about OSA properties.

It is known (see, e.g., [26]) that OSA can be reduced to MSFOL where subsorting is captured by *coercion functions* $c_s^{s'} \in F_{s,s'}$ defined for all $s \leq s'$. Intuitively, $c_s^{s'}$ denotes the injection from s to s'. However, this approach is not practically useful as noticed in [47, pp. 9]. We explain why by an example. Suppose there are three sorts $s \leq s' \leq s''$, a constant a of sort s, and a function $f \in F_{s'',s''}$. Then, $f(a)$ has two different parses when translated to MSFOL: $f(c_s^{s''}(a))$ and $f(c_{s'}^{s''}(c_s^{s'}(a)))$. Therefore, all OSA tools that are based on the reduction to MSFOL, need to reason *modulo* the triangle property $c_s^{s''}(a) = c_{s'}^{s''}(c_s^{s'}(a))$, which causes huge overhead. In contrast, our definition is more succinct and closer to the nature of OSA, because subsorting and overloading are naturally axiomatized, and no coercion functions are needed.

7 Instance: Parametric Sorts

Recall from Section 1.1 that $\mathbb{K}$ has built-in support for parametric sorts, e.g., parametric lists. In general, parametric sorts are difficult to define in a purely many-sorted setting, because they quickly lead to infinite theories with infinitely many sorts (e.g., base sort s, lists of s, lists of lists of s, etc.) and infinitely many, highly-homogeneous axioms for them.

On the other hand, AML gives us the flexibility to treat sorts as normal elements and to define *functions over sorts*, because there is no built-in distinction among elements, sorts, or functions. Therefore, we can easily extend the theory Γ^{MSFOL} to that of parametric sorts. As an example, we show how to axiomatize *parametric lists* as a finite theory Γ^{LIST} that extends Γ^{NAT}, the theory of natural numbers (Section 4.5). Let $\mathit{List} \in \Sigma^{\mathsf{LIST}}$ and axiomatize it as a function over sorts

$$\mathit{List}\colon \mathit{Sort} \to \mathit{Sort} \;\; /\!\!/ \;\; \text{(Function)}\; \forall s{:}\mathit{Sort} \,.\, \exists s'{:}\mathit{Sort} \,.\, \mathit{List}\, s = s'$$

This may look suspicious, as we quantify on s and s', which are sorts, but recall that the distinction between sorts and elements is only conceptual. AML itself makes no such distinction, so it is totally allowed to define *List* as above.

Parametric operations are also defined as functions. For example, *cons* takes x (of sort s) and l_1 (of sort $\mathit{List}\, s$), producing the list with head x and tail l_1. It is axiomatized by

$$\underbrace{\forall s{:}\mathit{Sort}\,.}_{\text{constrain the range of } s} \; \underbrace{\forall x{:}s \,.\, \forall l_1{:}\mathit{List}\, s \,.\, \exists l{:}\mathit{List}\, s \,.\, \mathit{cons}\, x\, l_1 = l}_{\text{specify that } \mathit{cons} \text{ is a function}}$$

The above is similar to the (Function) axiom (see Section 4) but uses s to range over *Sort*. For simplicity, we extend our function notation to also allow such "sort quantifiers":

$$\forall s{:}\mathit{Sort}\,.\; \mathit{cons}\colon s \times \mathit{List}\, s \to \mathit{List}\, s$$

Using this notation, we define *nil* and (parametric) *append*:

$$\forall s{:}Sort\,.\;\; nil : \rightarrow List\, s$$
$$\forall s{:}Sort\,.\;\; append : List\, s \times List\, s \rightarrow List\, s$$

As in Section 4.5 where we precisely define natural numbers as the smallest set built with 0 and *succ*, we can axiomatize the domain of *List s* to be the smallest set built with *nil* and *cons*, i.e., the inductive principle for finite lists:

$$\text{(Inductive List)} \quad \forall s{:}Sort\,.\,[\![List\ s]\!] = \mu L\,.\,nil \vee cons\ [\![s]\!]\ L$$

The above axiom supports inductive reasoning about lists. For example, assume a super-sort *Int* of *Nat*: $[\![Nat]\!] \subseteq [\![Int]\!]$. We can prove $[\![List\, Nat]\!] \subseteq [\![List\, Int]\!]$, using the AML proof system and (Inductive List) (see Proposition 33).

With parametric sorts, the *sort set Sort* also forms a term algebra built from base sorts such as *Nat* and *Int*, and the parametric sort *List*. This is axiomatized below:

$$\text{(Sorts)} \quad [\![Sort]\!] = \mu S\,.\,Nat \vee Int \vee List\ S$$

We can also allow only primitive lists and forbid nested lists:

$$\text{(No Nested Lists)} \quad [\![Sort]\!] = Nat \vee Int \vee List\ Nat \vee List\ Int$$

A more modular way is to introduce a new symbol *PrimitiveSort*, axiomatize it by $PrimitiveSort = Nat \vee Int$, and let *Sort* contain only the primitive sorts and lists over them:

$$Sort = PrimitiveSort \vee List\ PrimitiveSort$$

In the above, we overloaded *nil* over all lists. We can also make it parametric, by replacing $nil : \rightarrow List\ s$ with:

$$\text{(Parametric Nil)} \quad \forall s{:}Sort\,.\,\exists l{:}List\, s\,.\,nil\ s = l$$

(Inductive List) is modified accordingly:

$$\forall s{:}Sort\,.\,[\![List\ s]\!] = \mu L\,.\,nil\ s \vee cons\ [\![s]\!]\ L \quad /\!/ \text{ parametric } nil$$

The main message is that there are no hard axioms or definitions, and AML offers much extensibility and flexibility.

8 Putting Them Together: Semantics of $\mathbb{K}$

Here we complete our agenda of formalizing $\mathbb{K}$'s features as AML axioms and notations. We show that $\mathbb{K}$ definitions are AML theories, and that $\mathbb{K}$'s execution and verification tools are best effort implementations (heuristics) of AML proof search. Hence, AML gives a unifying logical foundation for $\mathbb{K}$.

For concreteness, we take IMP and its $\mathbb{K}$ definition in Fig. 2 as an example. We consider a simple (symbolic) IMP program, `SUM ≡ int n,s; n=N:Int; while(n){s=s+n;n=n-1;}`, which computes the sum from 1 to the (symbolic) input `N:Int`. Let `LOOP` be the while loop of `SUM`.

In the following, we derive from the $\mathbb{K}$ definition of IMP (Fig. 2) an AML signature Σ^{IMP} and a theory Γ^{IMP} that defines IMP syntax and semantics. Then we use AML reasoning within Γ^{IMP} to formalize program execution and verification for IMP. For simplicity, we use L*m* to mean line *m* in Fig. 2.

IMP syntax and configurations. $\mathbb{K}$ sorts (i.e., nonterminals) such as `Exp`, `Int`, `Id` define corresponding sorts in AML: *Exp*, *Int*, *Id* ($\in$ *Sort*). A production rule either defines a subsort relation or a many-/order-sorted function. For example, production rule for `Int` in L2 defines the subsorting axiom $[\![Int]\!] \subseteq [\![Exp]\!]$; production rule for `Exp "+" Exp` in L2 defines a function $plusExp\colon Exp \times Exp \to Exp$. The parametric sort `Ids` in L13 is defined by the parametric sort *List Id* in Section 7.

Program configurations can be defined similarly to the concrete syntax. Every $\mathbb{K}$ *cell*, such as `<k/>`, defines a sort $kCell \in Sort$ and a function $k\colon Pgm \to kCell$. Similarly, $\mathbb{K}$ cells `<state/>` and `<T/>` define the sorts *stateCell* and *TCell* and functions $state\colon Map \to stateCell$ and $T\colon kCell \times stateCell \to TCell$, where *Map* is a predefined sort for finite maps (see, e.g., [53, Section 9.2]). We often write *TCell* as *Cfg*, denoting the sort of top-level program configurations.

Rewrite rules. $\mathbb{K}$ defines language semantics in terms of *rewrite rules* of the form $\varphi_{lhs} \Rightarrow \varphi_{rhs}$, where φ_{lhs} and φ_{rhs} are configuration patterns of sort *Cfg*, i.e., $\varphi_{lhs}, \varphi_{rhs} \subseteq [\![Cfg]\!]$. These rewrite rules yield a *transition system* over configurations as the formal language semantics, so we first capture the transition relation in AML. Inspired by temporal logic, we define a symbol $\bullet \in \Sigma^{\mathsf{IMP}}$, called *one-path next*, and its arity axiom $\bullet\,[\![Cfg]\!] \subseteq [\![Cfg]\!]$. Intuitively, $\bullet\gamma$ contains configurations γ' such that γ is a next state of γ'. Then, rule $\varphi_{lhs} \Rightarrow \varphi_{rhs}$ defines an axiom (Semantics) $\varphi_{lhs} \to \bullet\,\varphi_{rhs}$, which states that any configuration matching φ_{lhs} has *at least one* next state matching φ_{rhs}.

Program execution. One-path next $\bullet \in \Sigma^{\mathsf{IMP}}$ only defines one-step rewrites; e.g., $\varphi \to \bullet\psi$ means φ rewrites to ψ in one step; $\varphi \to \bullet\bullet\,\psi$ means φ rewrites to ψ in two steps; etc. However, program execution can take arbitrarily, finitely many steps, so we need a pattern to denote that φ rewrites to ψ in any *finitely many* steps. Such finiteness properties are known to be outside of the scope of FOL, because fixpoints are needed. AML has built-in support for fixpoints, so it can define such finiteness properties, such as the eventually operator $\Diamond\varphi \equiv \mu X.\varphi \vee \bullet X$. Then, we can use $\varphi \to \Diamond\psi$ to mean program executes from φ to ψ. As a concrete example, consider executing the `SUM` program from the configuration:

$$\Psi(\texttt{SUM}, n, s) \equiv T\,(k\,\texttt{SUM})\,(state\,(merge\,(mapsto\,\texttt{n}\,n)(mapsto\,\texttt{s}\,s)))$$

whose computation is `code` and whose state maps `n` to n and `s` to s. Here, *merge* and *mapsto* are predefined map functions; see, e.g., [53, Section 9.2]. Then, $\varphi_0 \equiv \Psi(\texttt{LOOP}, 100, 0)$ denotes the configuration that reaches the loop with $n = 100$. The following AML proof steps formalize the first few execution steps that $\mathbb{K}$ applies, until we reach the loop again:

$$\Gamma^{\mathsf{IMP}} \vdash \varphi_0 \to \bullet\varphi_1 \text{ where } \varphi_1 \equiv \Psi(\texttt{if(n)\{s=s+n;n=n-1;LOOP\}\{\}}, 100, 0)$$
$$\Gamma^{\mathsf{IMP}} \vdash \varphi_1 \to \bullet\varphi_2 \text{ where } \varphi_2 \equiv \Psi(\texttt{if(100)\{s=s+n;n=n-1;LOOP\}\{\}}, 100, 0)$$
$$\Gamma^{\mathsf{IMP}} \vdash \varphi_2 \to \bullet\varphi_3 \text{ where } \varphi_2 \equiv \Psi(\texttt{s=s+n;n=n-1;LOOP}, 100, 0)$$
$$\Gamma^{\mathsf{IMP}} \vdash \cdots$$
$$\Gamma^{\mathsf{IMP}} \vdash \cdots \to \bullet\varphi_{12} \text{ where } \varphi_{12} \equiv \Psi(\texttt{LOOP}, 99, 100)$$

By AML proof rule (Framing), we have $\Gamma^{\mathsf{IMP}} \vdash \bullet\varphi_1 \to \bullet\bullet\varphi_2$, $\Gamma^{\mathsf{IMP}} \vdash \bullet\bullet\,\varphi_2 \to \bullet\bullet\bullet\,\varphi_3$, etc. We can "chain" them together by FOL reasoning and prove $\Gamma^{\mathsf{IMP}} \vdash \varphi_0 \to \underbrace{\bullet\cdots\bullet}_{\text{12 times}}\varphi_{12}$. Finally, by applying fixpoint reasoning and unfolding $\Diamond\varphi_{12}$ 12 times, we prove $\Gamma^{\mathsf{IMP}} \vdash \varphi_0 \to \Diamond\varphi_{12}$. This is

only an execution fragment. For full execution, we continue the process until we reach final configuration $\varphi_m \equiv \Psi(.\,, 0, 5050)$ (where m is the number of execution steps and "." denotes the empty computation, i.e., termination) and prove $\Gamma^{\mathsf{IMP}} \vdash \varphi_0 \to \Diamond\varphi_M$.

In conclusion, $\mathbb{K}$'s program execution tool becomes best effort implementation of proof searching for AML theorems $\Gamma^{\mathsf{IMP}} \vdash \varphi_0 \to \Diamond\varphi_M$. $\mathbb{K}$'s debugger tool allows users to step through and visualize executions of any programs in any programming languages that have a $\mathbb{K}$ semantics.

More dynamic patterns. Many dynamic properties and modal operators can be defined as AML patterns. Here we define some modal operators necessary for program verification. "All-path next" $\bigcirc\varphi \equiv \neg_{Cfg} \bullet \neg_{Cfg}\varphi$ is the dual of one-path next; it is matched by states whose next states *all* match φ. Here, $\neg_{Cfg}\varphi \equiv \neg\varphi \wedge [\![Cfg]\!]$ is matched by all *configurations* not matching φ. "Always" $\Box\varphi \equiv \neg_{Cfg}\Diamond\neg_{Cfg}\varphi$ is the dual of "eventually", matched by all states where φ holds now and hereafter. "Well-founded states" $\mathsf{WF} \equiv \mu X.\, \bigcirc\Box X$ is matched by all well-founded states, i.e., those which do not have infinite execution traces.

Program verification. Verification is different from execution, in that it considers *partial correctness*, i.e., the post-condition φ_{post} needs to be satisfied only *on termination*. Therefore, we weaken the definition of $\Diamond\varphi$ that is used to formalize execution, to $\Diamond_w\varphi \equiv (\neg_{Cfg}\mathsf{WF}) \vee \Diamond\varphi$, called *weak-eventually*, and use $\varphi_{pre} \to \Diamond_w\varphi_{post}$ to mean that φ_{pre} either diverges (i.e. has an infinite execution trace) or reaches φ_{post} eventually, capturing precisely partial correctness semantics.

As a concrete example, we use AML to formalize the verification task of the *invariant claim* of the while loop:

$$\Phi_{inv} \equiv \forall n{:}Int.\, \forall s{:}Int.\, (\Psi(\texttt{LOOP}, n, s) \wedge n \geq 0) \to \Diamond_w\Psi(.\,, 0, s + n(n+1)/2)$$

Intuitively, it says that if `LOOP` is executed at where `n` is n and `s` is s, then on termination `n` becomes 0 and `s` becomes $s+n(n+1)/2$. $\mathbb{K}$'s verification tool is based on *circular reasoning* [54]; that is, $\mathbb{K}$ moves Φ_{inv} to a set of *circularities* and starts executing `LOOP` *symbolically*. If $n = 0$, the program terminates and verification finishes. For $n \geq 1$, $\mathbb{K}$ consumes the loop body and reaches $\Psi(\texttt{LOOP}, n-1, s+n) \wedge n \geq 1$. Then it uses Φ_{inv} as a *normal semantic axiom* to finish the proof by instantiating n to $n-1$ and s to $s+n$.

We formalize the above circular reasoning as AML fixpoint reasoning; see Appendix J for details. The key observation is that, by partial correctness, we only need to prove Φ_{inv} for *well-founded* states; i.e., $\Gamma^{\mathsf{IMP}} \vdash \Phi_{inv}$ iff $\Gamma^{\mathsf{IMP}} \vdash \mathsf{WF} \to \Phi_{inv}$, where $\mathsf{WF} \equiv \mu X.\, \bigcirc\Box X$ denotes well-founded states as defined earlier. Then by (Knaster-Tarski), it suffices to prove $\Gamma^{\mathsf{IMP}} \vdash \bigcirc\Box\Phi_{inv} \to \Phi_{inv}$ with $\bigcirc\Box\Phi_{inv}$ being an additional condition, saying that Φ_{inv} holds after any rewrite step, thus yielding circular reasoning. In conclusion, $\mathbb{K}$'s program verification tool becomes a best effort implementation of proof searching for AML theorems $\Gamma^{\mathsf{IMP}} \vdash \varphi_{pre} \to \Diamond_w\varphi_{post}$.

9 New $\mathbb{K}$ Features Based on AML

We propose two new features for $\mathbb{K}$, frequently requested by users, based on AML. They are nontrivial extensions of existing features but can be uniformly treated in AML and thus can correctly extend existing $\mathbb{K}$ implementations. We use them to demonstrate the benefit of using a logic like AML as the logical foundation and formal semantics of $\mathbb{K}$.

9.1 Feature A: Function Sorts/Types

Our first feature proposal is *function sorts/types*. We define $\mathit{Function} : \mathit{Sort} \times \mathit{Sort} \to \mathit{Sort}$ to be a sort constructor (like *List* in Section 7) and use $\mathit{Function}\ s\ s'$ to denote the sort of functions from s to s', as axiomatized below:

(Function Sort)
$$\forall s{:}\mathit{Sort}\,.\,\forall s'{:}\mathit{Sort}\,.\,[\![\mathit{Function}\ s\ s']\!] = \exists f\,.\,(f \wedge \forall x{:}s\,.\,\exists y{:}s'\,.\,f\,x = y)$$

Recall that $\exists$ means set union (see Definition 3). Therefore, $[\![\mathit{Function}\ s\ s']\!]$ is matched by all f such that $\forall x{:}s\,.\,\exists y{:}s'.\,f\,x = y$, i.e., f is a function from s to s'. Multiary functions can be defined as usual making use of currying, as follows (quantification over sorts is defined in Section 7):

$$\begin{aligned}&\mathit{Function}\ s_1\ \cdots\ s_n\ s\\ &\quad\equiv \mathit{Function}\ s_1\ (\mathit{Function}\ s_2\ \cdots\ (\mathit{Function}\ s_n\ s)\cdots)\end{aligned}$$

Function sorts are useful to formalize *functional languages* such as OCaml [37] and Haskell [38] in $\mathbb{K}$, where functions are first-class citizens and can take functions as arguments. For example, we extend Γ^{LIST} with two higher-order functions *fold* and *map* as follows:

$$\begin{array}{|l}\hline
\forall s{:}\mathit{Sort}\,.\,\forall s'{:}\mathit{Sort}\,.\\
\quad \mathit{fold}{:}\ \mathit{Function}\ s'\ s\ s' \times s' \times \mathit{List}\ s \to s'\\
\forall f{:}\mathit{Function}\ s'\ s\ s'\,.\,\forall x{:}s'\,.\ \mathit{fold}\ f\ x\ \mathit{nil} = x\\
\forall f{:}\mathit{Function}\ s'\ s\ s'\,.\,\forall x{:}s'\,.\,\forall y{:}s\,.\,\forall l{:}\mathit{List}\ s\,.\\
\quad \mathit{fold}\ f\ x\ (\mathit{cons}\ y\ l) = \mathit{fold}\ f\ (f\,x\,y)\ l\\
\hline
\forall s{:}\mathit{Sort}\,.\,\forall s'{:}\mathit{Sort}\,.\\
\quad \mathit{map}{:}\ \mathit{Function}\ s\ s' \times \mathit{List}\ s \to \mathit{List}\ s'\\
\forall g{:}\mathit{Function}\ s\ s'\,.\ \mathit{map}\ g\ \mathit{nil} = \mathit{nil}\\
\forall g{:}\mathit{Function}\ s\ s'\,.\,\forall y{:}s\,.\,\forall l{:}\mathit{List}\ s\,.\\
\quad \mathit{map}\ g\ (\mathit{cons}\ y\ l) = \mathit{cons}\ (g\ y)\ (\mathit{map}\ g\ l)\\
\hline\end{array}$$

We propose the $\mathbb{K}$ team to also incorporate conventional sort/type inference procedures in future versions of $\mathbb{K}$, because with those, all the non-gray parts above can be inferred, in which case users would only write the gray equations, making defining semantics in $\mathbb{K}$ closely resemble functional programming in languages like Haskell or OCaml.

9.2 Feature B: Dependent Sorts/Types

Our second feature proposal is *dependent sorts (types)*, which are like parametric sorts but whose parameters are elements, i.e., arbitrary data, instead of sorts. Since AML makes no distinction between elements and sorts, they are defined uniformly by patterns, and it is straightforward to define dependent sorts, following the definition of parametric lists in Section 7.

As an example, we define a dependent sort *MInt* for *machine integers*; for $n \geq 1$, we let $\mathit{MInt}\ n$ denote the sort of machine integers of *size* n, i.e., natural numbers less than 2^n. For clarity, we define a new sort *Size* for positive naturals and define $\mathit{MInt} : \mathit{Size} \to \mathit{Sort}$ as a function as below:

$$[\![Size]\!] = succ\ [\![Nat]\!]$$
$$\forall n{:}Size\,.\,[\![MInt\ n]\!] = \exists x{:}Nat\,.\,x \wedge x < pow_2\ n$$

where $pow_2 \colon Nat \to Nat$ (power of 2) and $\_<\_$ are defined as usual. Then we define functions over machine integers, such as *mplus* and *mmult*, by defining their arities and reusing natural numbers addition *plus* and multiplication *mult*:

$$\forall n{:}Size\,.\,mplus \colon MInt\ n \times MInt\ n \to MInt\ (succ\ n)$$
$$\forall n{:}Size\,.\,\forall x{:}MInt\ n\,.\,\forall y{:}MInt\ n\,.\,mplus\ x\ y = plus\ x\ y$$
$$\forall n{:}Size\,.\,\forall m{:}Size\,.\,mmult \colon MInt\ n \times MInt\ m \to MInt\ (plus\ n\ m)$$
$$\forall n{:}Sizeldot\forall m{:}Size\,.\,\forall x{:}MInt\ n\,.\,\forall y{:}MInt\ m\,.\,mmult\ x\ y = mult\ x\ y$$

10 Related Work

Other theoretical attempts to formalizing $\mathbb{K}$. These include the double-pushout graph transformations [56], an Isabelle definition [39], and a Coq definition based on co-induction [45], all of which require heavy theory transformations with a big representational distance from the original definitions, and are targeted at only a subset of $\mathbb{K}$ features such as reachability, partial correctness, or coinduction, while with AML we aim at giving full formal semantics to $\mathbb{K}$.

Reachability logic [54, 15] and matching logic [53, 9] are a series of research aiming at formalizing $\mathbb{K}$'s program verification tools and are what inspired AML. Reachability logic has built-in support for reasoning about reachability properties, and matching logic (which is an MSFOL variant; see Section 5) has built-in support for defining many-sorted structures, making them over-complicated and less flexible to be extended to deal with more complex dynamic properties and sort structures (e.g., ordered sorts, parametric sorts) needed by $\mathbb{K}$. In contrast, AML adopts a *minimalist design* and defines only the simplest building blocks, with which more complex and notationally heavier concepts can be axiomatized as theories, as we have shown in this paper. We think AML is more suitable to be taken by the $\mathbb{K}$ community to be the foundation of $\mathbb{K}$, due to its simplicity, minimality, and high flexibility that make it easier to formalize existing features and develop new features, especially at this stage where $\mathbb{K}$ still seems to be continuously evolving and changing.

Other language frameworks. There are several other important language frameworks besides $\mathbb{K}$ for defining formal semantics of programming languages. *CENTAUR* [6] is one of the earliest systems that take formal language definitions and automatically generate programming environments with language tools such as interpreters and debuggers, equipped with graphic interfaces. *Proof assistants* such as Coq [3] and Isabelle [48] have been extensively used as language frameworks, where program verification is framed as proving theorems, mostly interactively with remarkable human effort, although (semi-)automation is available in some cases. Lightweight tools such as *Ott* [57] provide an intuitive frontend language to define formal syntax and semantics, accompanied with automatic tools that sanity-check definitions and translate them to proof assistants where proofs are done.

Our goal with AML is different. We aimed at a direct and minimal logical foundation for $\mathbb{K}$, without translation to other frameworks or logics, because such a translation often yields heavy notational cost and, more importantly, causes difficulties to translate back (i.e., interpret) failed verification attempts, error messages, etc. As discussed in Section 2.1, the syntax of AML is

minimal, where every logical construct is critical and has a unique semantic meaning. The 13-rule proof system of AML is succinct and compact, and it support all essential logical reasoning (e.g., FOL, frame, fixpoint, and equational reasoning) that is required in program analysis and verification.

There are several language frameworks that are also *rewriting-based*, like $\mathbb{K}$. *Component-based specification* [4] predefines a set of fundamental language constructs and uses them to define languages modularly. *Spoofax* [7] is a platform for designing *domain specific languages* (DSL), integrated with various language tools such as SDF [61] for formal syntax, Stratego [62] for code generation, FlowSpec [59] for data flow analysis, etc. *PLT Redex* [21] is a DSL for designing rewrite-based operational semantics, fully integrated within a target programming language such as Scheme or Racket.

Since $\mathbb{K}$ is also rewriting-based and supports all the features of the above frameworks, AML can be a proper logical foundation for the above-mentioned frameworks, too, so they can share the same simple yet powerful AML prover and proof checkers to ensure semantic correctness of the claims they make, no matter what heuristics or algorithms they employed.

11 Conclusion

Previously, the development and implementation of $\mathbb{K}$ seemed to be mostly driven by practical needs from user requests instead of a clear and solid mathematical foundation. This resulted in some ad hoc engineering hacks in existing $\mathbb{K}$ tools and uncertainties about the *formal semantics* of $\mathbb{K}$.

In this paper, we proposed *applicative matching logic* (AML) as what we believe is the right logical foundation and formal semantics of $\mathbb{K}$. Specifically, we took the major logical frameworks that are fully or partially supported by $\mathbb{K}$ implementations and showed how they can be captured as theories and notations in AML. We put all results together and showed from a theoretical point of view that $\mathbb{K}$'s program execution and verification tools are best effort implementations in realizing AML reasoning and proof searching. Finally, we proposed two nontrivial extensions of $\mathbb{K}$ features and showed that AML offers a natural treatment for both, showing additional benefits of adopting a logic like AML as the foundation of $\mathbb{K}$.

References

[1] Astesiano, E., Bidoit, M., Kirchner, H., Krieg-Brückner, B., Mosses, P.D., Sannella, D., Tarlecki, A.: CASL: the common algebraic specification language. Theoretical Computer Science **286**(2), 153–196 (2002). https://doi.org/10.1016/S0304-3975(01)00368-1, current trends in Algebraic Development Techniques

[2] Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11). pp. 171–177. Springer (2011)

[3] Bertot, Y., Castran, P.: Interactive theorem proving and program development: Coq'Art the calculus of inductive constructions. Springer (2010)

[4] van Binsbergen, L.T., Sculthorpe, N., Mosses, P.D.: Tool support for component-based semantics. In: Companion Proceedings of the 15th International Conference on Modularity. pp. 8–11. ACM (2016)

[5] Bogdănaş, D., Roşu, G.: K-Java: A complete semantics of Java. In: Proceedings of the 42nd Symposium on Principles of Programming Languages (POPL'15). pp. 445–456. ACM (2015)

[6] Borras, P., Clément, D., Despeyroux, T., Incerpi, J., Kahn, G., Lang, B., Pascual, V.: CENTAUR: The system. In: Proceedings of the 3rd ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments (SDE'88). pp. 14–24. ACM (1988)

[7] van den Brand, M., Heering, J., Klint, P., Olivier, P.A.: Compiling language definitions: The ASF+SDF compiler. ACM Transactions on Programming Languages and Systems (TOPLAS'02) **24**(4), 334–368 (2002)

[8] Burstall, R.M., Goguen, J.A.: Putting theories together to make specifications. In: IJCAI'77. pp. 1045–1058. Morgan Kaufmann Publishers Inc. (1977), http://dl.acm.org/citation.cfm?id=1622943.1623045

[9] Chen, X., Roşu, G.: Matching μ-logic. In: Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'19) (2019)

[10] Chen, X., Roşu, G.: Matching μ-logic. Tech. rep., University of Illinois at Urbana-Champaign (2019), http://hdl.handle.net/2142/102281

[11] Chen, X., Roşu, G.: A language-independent program verification framework. In: Proceedings of the 8th International Symposium on Leveraging Applications of Formal Methods (ISoLA'18). vol. 11245, pp. 92–102. Springer (2018)

[12] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martı-Oliet, N., Meseguer, J., Quesada, J.: Maude: specification and programming in rewriting logic. Theoretical Computer Science **285**(2), 187–243 (2002)

[13] Cohn, A.: Taxonomic reasoning with many-sorted logics. Artificial Intelligence Review **3**(2), 89–128 (1989)

[14] Cohn, A.G.: A more expressive formulation of many sorted logic. Journal of Automated Reasoning **3**(2), 113–200 (1987). https://doi.org/10.1007/BF00243207

[15] Ştefănescu, A., Ciobâcă, c., Mereuţă, R., Moore, B.M., Şerbănuţă, T.F., Roşu, G.: All-path reachability logic. In: Proceedings of the Joint 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications (RTA-TLCA'14). vol. 8560, pp. 425–440. Springer (2014)

[16] Dasgupta, S., Park, D., Kasampalis, T., Adve, V.S., Roşu, G.: 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). ACM (2019)

[17] De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08). pp. 337–340. Springer (2008)

[18] Diaconescu, R., Futatsugi, K.: CafeOBJ report, The language, proof techniques, and methodologies for object-oriented algebraic specification, vol. 6. World Scientific (1998)

[19] Ésik, Z.: Completeness of Park induction. Theoretical Computer Science **177**(1), 217–283 (1997)

[20] Farmer, W.M., Guttman, J.D.: A set theory with support for partial functions. Studia Logica **66**(1), 59–78 (2000)

[21] Felleisen, M., Findler, R.B., Flatt, M.: Semantics engineering with PLT Redex. Mit Press (2009)

[22] Gödel, K.: On formally undecidable propositions of principia Mathematica and related systems. Courier corporation (1992)

[23] Goguen, J., Winkler, T., Meseguer, J., Futatsugi, K., Jouannaud, J.P.: Introducing OBJ. In: Software Engineering with OBJ: Algebraic specification in action, pp. 3–167. Kluwer (2000)

[24] Goguen, J.A., Thatcher, J.W., Wagner, E.G.: An initial algebra approach to the specification, correctness, and implementation of abstract data types. IBM research reports, IBM Research Center (1976)

[25] Goguen, J., Diaconescu, R.: An Oxford survey of order sorted algebra. Mathematical Structures in Computer Science **4**(3), 363–392 (1994). https://doi.org/10.1017/S0960129500000517

[26] Goguen, J., Meseguer, J.: Order-sorted algebra, Part 1: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theoretical Computer Science **105**(2), 217–273 (1992)

[27] Goguen, J., Thatcher, J., Wagner, E., Wright, J.: Initial algebra semantics and continuous algebras. Journal of the ACM **24**(1), 68–95 (1977)

[28] Goguen, J., Winkler, T., Meseguer, J., Futatsugi, K., Jouannaud, J.P.: Software engineering with OBJ: Algebraic specification in action, chap. Introducing OBJ, pp. 3–167. Springer (2000)

[29] Gurevich, Y., Shelah, S.: Fixed-point extensions of first-order logic. In: Proceedings of the 26th Annual Symposium on Foundations of Computer Science (SFCS'85). pp. 346–353. IEEE (1985)

[30] Hathhorn, C., Ellison, C., Roşu, G.: Defining the undefinedness of C. In: Proceedings of the 36th annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15). pp. 336–345. ACM (2015)

[31] Hennicker, R., Wirsing, M.: Proof systems for structured algebraic specifications: An overview. In: Chlebus, B.S., Czaja, L. (eds.) Fundamentals of Computation Theory. pp. 19–37. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)

[32] Hildenbrandt, E., Saxena, M., Zhu, X., Rodrigues, N., Daian, P., Guth, D., Moore, B., Zhang, Y., Park, D., Ştefănescu, A., Roşu, G.: KEVM: A complete semantics of the Ethereum virtual machine. In: Proceedings of the 2018 IEEE Computer Security Foundations Symposium (CSF'18). IEEE (2018), `http://jellopaper.org`

[33] K Development Team: K framework tools 5.0. `https://github.com/kframework/k` (Jan 2020)

[34] K Development Team: The Kore language. `https://github.com/kframework/kore` (Jan 2020)

[35] Kasampalis, T., Guth, D., Moore, B., Şerbănuţă, T.F., Zhang, Y., Filaretti, D., Şerbănuţă, V., Johnson, R., Roşu, G.: IELE: A rigorously designed language and tool ecosystem for the blockchain. In: Proceeding of the 23rd International Symposium on Formal Methods (FM'19) (2019)

[36] Kozen, D.: Results on the propositional μ-calculus. In: Proceedings of the 9th International Colloquium on Automata, Languages and Programming (ICALP'82). pp. 348–359. Springer (1982)

[37] Lazar, D.: Formal semantics of OCaml (2013), `https://github.com/davidlazar/ocaml-semantics`

[38] Lazar, D.: Formal semantics of Haskell (2014), `https://github.com/davidlazar/haskell-semantics`

[39] Li, L., Gunter, E.: IsaK-static A complete static semantics of K. In: Formal Aspects of Component Software. pp. 196–215. Springer (2018)

[40] Liskov, B., Zilles, S.: Programming with abstract data types. SIGPLAN Not. **9**(4), 50–59 (1974). https://doi.org/10.1145/942572.807045

[41] Malc'ev, A.I.: Axiomatizable classes of locally free algebras of various type. The Metamathematics of Algebraic Systems: Collected Papers pp. 262–281 (1936)

[42] Meinke, K., Tucker, J.V. (eds.): Many-sorted logic and its applications. John Wiley & Sons, Inc. (1993)

[43] Mendelson, E.: Introduction to mathematical logic. Springer (1979)

[44] Meseguer, J., Goguen, J.: Order-sorted algebra solves the constructor-selector, multiple representation, and coercion problems. Information and Computation **103**(1), 114–158 (1993)

[45] Moore, B., Peña, L., Roşu, G.: Program verification by coinduction. In: Proceedings of the 27th European Symposium on Programming (ESOP'18). pp. 589–618. Springer (2018)

[46] Mosses, P.D.: CASL reference manual, The complete documentation of the common algebraic specification language, vol. 2960. Springer (2004)

[47] Nelson, T., Dougherty, D., Fisler, K., Krishnamurthi, S.: On the finite model property in order-sorted logic. Tech. rep., Worcester Polytechnic Institute (2010)

[48] Nipkow, T., Wenzel, M., Paulson, L.: Isabelle/HOL: A proof assistant for higher-order logic. Springer (2002)

[49] Park, D., Ştefănescu, A., Roşu, G.: KJS: A complete formal semantics of JavaScript. In: Proceedings of the 36th annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15). pp. 346–356. ACM (2015)

[50] Peano, G.: Arithmetices principia: Nova methodo exposita. Fratres Bocca (1889)

[51] Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS'77). pp. 46–57. IEEE (1977)

[52] Poigné, A.: Once more on order-sorted algebras. In: Tarlecki, A. (ed.) Mathematical Foundations of Computer Science 1991. pp. 397–405. Springer Berlin Heidelberg, Berlin, Heidelberg (1991)

[53] Roşu, G.: Matching logic. Logical Methods in Computer Science **13**(4), 1–61 (2017)

[54] Roşu, G., Ştefănescu, A., Ciobâcă, Ş., Moore, B.M.: One-path reachability logic. In: Proceedings of the 28th Symposium on Logic in Computer Science (LICS'13). pp. 358–367. IEEE (2013)

[55] Rosu, G.: K—A semantic framework for programming languages and formal analysis tools. In: Dependable Software Systems Engineering. IOS Press (2017)

[56] Şerbănuţă, T.F., Roşu, G.: A truly concurrent semantics for the K framework based on graph transformations. In: Proceedings of the 6th International Conference on Graph Transformation (ICGT'12). pp. 294–310. Springer (2012)

[57] Sewell, P., Nardelli, F.Z., Owens, S., Peskine, G., Ridge, T., Sarkar, S., Strniša, R.: Ott: Effective tool support for the working semanticist. Journal of Functional Programming **20**(1), 71–122 (2010)

[58] Shoenfield, J.R.: Mathematical logic. Addison-Wesley Pub. Co (1967)

[59] Smits, J., Visser, E.: FlowSpec: Declarative dataflow analysis specification. In: Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering (SLE'17). pp. 221–231. ACM (2017)

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

[61] Visser, E.: Syntax definition for language prototyping. Ph.D. thesis, University of Amsterdam (1997)

[62] Visser, E., Benaissa, Z.e.A., Tolmach, A.: Building program optimizers with rewriting strategies. In: Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming (ICFP'98). pp. 13–26. ACM (1998)

A Knaster-Tarski Fixpoint Theorem

Theorem 19 (Knaster-Tarski [60]). *For any nonempty set M and any monotone function $\mathcal{F} : \mathcal{P}(M) \to \mathcal{P}(M)$ with $\mathcal{F}(A) \subseteq \mathcal{F}(B)$ for all $A \subseteq B$, where $\mathcal{P}(M)$ denotes the powerset of M, we have that $\mathcal{F}$ has a unique least fixpoint $\mu\mathcal{F}$ and a unique greatest fixpoint $\nu\mathcal{F}$, given as:*

$$\mu\mathcal{F} = \bigcap\{A \subseteq M \mid \mathcal{F}(A) \subseteq A\} \qquad\qquad \nu\mathcal{F} = \bigcup\{A \subseteq M \mid A \subseteq \mathcal{F}(A)\}$$

B Proofs of results in Sections 2

Here we prove all results in Section 2.

Proof of Proposition 4. Simple, by applying definitions directly.

(Case $\neg\varphi$): $\bar{\rho}(\neg\varphi) = \bar{\rho}(\varphi \to \bot) = M \setminus (\bar{\rho}(\varphi) \setminus \bar{\rho}(\bot)) = M \setminus (\bar{\rho}(\varphi) \setminus \emptyset) = M \setminus \bar{\rho}(\varphi)$.

(Case $\varphi_1 \vee \varphi_2$): $\bar{\rho}(\varphi_1 \vee \varphi_2) = \bar{\rho}(\neg\varphi_1 \to \varphi_2) = M \setminus (\bar{\rho}(\neg\varphi_1) \setminus \bar{\rho}(\varphi_2)) = M \setminus ((M \setminus \bar{\rho}(\varphi_1)) \setminus \bar{\rho}(\varphi_2)) = \bar{\rho}(\varphi_1) \cup \bar{\rho}(\varphi_2)$.

(Case $\varphi_1 \wedge \varphi_2$): $\bar{\rho}(\varphi_1 \wedge \varphi_2) = \bar{\rho}(\neg\varphi_1 \vee \neg\varphi_2) = \bar{\rho}(\neg\varphi_1) \vee \bar{\rho}(\neg\varphi_2) = (M \setminus \bar{\rho}(\varphi_1)) \vee (M \setminus \bar{\rho}(\varphi_2)) = \bar{\rho}(\varphi_1) \wedge \bar{\rho}(\varphi_2)$.

(Case $\top$): $\bar{\rho}(\top) = \bar{\rho}(\neg\bot) = M \setminus \bar{\rho}(\bot) = M \setminus \emptyset = M$.

(Case $\varphi_1 \leftrightarrow \varphi_2$): $\bar{\rho}(\varphi_1 \leftrightarrow \varphi_2) = \bar{\rho}((\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)) = \bar{\rho}(\varphi_1 \to \varphi_2) \cap \bar{\rho}(\varphi_2 \to \varphi_1) = (M \setminus (\bar{\rho}(\varphi_1) \setminus \bar{\rho}(\varphi_2))) \cap (M \setminus (\bar{\rho}(\varphi_2) \setminus \bar{\rho}(\varphi_1))) = M \setminus (\bar{\rho}(\varphi_1) \vartriangle \bar{\rho}(\varphi_2))$.

(Case $\forall x.\varphi$): $\bar{\rho}(\forall x.\varphi) = \bar{\rho}(\neg\exists x.\neg\varphi) = M \setminus \bar{\rho}(\exists x.\neg\varphi) = M \setminus \bigcup_{a\in M} \overline{\rho[a/x]}(\neg\varphi) = M \setminus \bigcup_{a\in M} \left(M \setminus \overline{\rho[a/x]}(\varphi)\right) = \bigcap_{a\in M} \overline{\rho[a/x]}(\varphi)$.

(Case $\nu X.\varphi$): $\bar{\rho}(\nu X.\varphi) = \bar{\rho}(\neg\mu X.\neg\varphi[\neg X/X]) = M \setminus \bar{\rho}(\mu X.\neg\varphi[\neg X/X]) = M \setminus \mu\mathcal{F}^{\rho}_{\neg\varphi[\neg X/X],X}$. Note that $\mu\mathcal{F}^{\rho}_{\neg\varphi[\neg X/X]} = \bigcup\{A \subseteq M \mid \mathcal{F}^{\rho}_{\neg\varphi[\neg X/X]}(A) \subseteq A\} = \bigcup\{A \subseteq M \mid \overline{\rho[A/X]}(\neg\varphi[\neg X/X]) \subseteq A\}$. Then $M \setminus \mu\mathcal{F}^{\rho}_{\neg\varphi[\neg X/X],X} = M \setminus \bigcup\{A \subseteq M \mid \overline{\rho[A/X]}(\neg\varphi[\neg X/X])\} = \bigcap\{M \setminus A \mid \overline{\rho[A/X]}(\neg\varphi[\neg X/X]) \subseteq A\} = \bigcap\{B \mid \overline{\rho[B/X]}(\neg\varphi) \subseteq (M \setminus B)\} = \bigcap\{B \mid \overline{\rho[B/X]}(\varphi) \supseteq B\} = \nu\mathcal{F}^{\rho}_{\varphi,X}$ □

Proposition 20. With the above notation, the following hold:

- $\bar{\rho}(\lceil\varphi\rceil) = M$ iff $\bar{\rho}(\varphi) \neq \emptyset$; and $\bar{\rho}(\lceil\varphi\rceil) = \emptyset$ iff $\bar{\rho}(\varphi) = \emptyset$;
- $\bar{\rho}(\lfloor\varphi\rfloor) = M$ iff $\bar{\rho}(\varphi) = M$; and $\bar{\rho}(\lfloor\varphi\rfloor) = \emptyset$ iff $\bar{\rho}(\varphi) \neq \emptyset$;
- $\bar{\rho}(\varphi_1 = \varphi_2) = M$ iff $\bar{\rho}(\varphi_1) = \bar{\rho}(\varphi_2)$; and $\bar{\rho}(\varphi_1 = \varphi_2) = \emptyset$ iff $\bar{\rho}(\varphi_1) \neq \bar{\rho}(\varphi_2)$;
- $\bar{\rho}(x \in \varphi) = M$ iff $\rho(x) \in \bar{\rho}(\varphi)$; and $\bar{\rho}(x \in \varphi) = \emptyset$ iff $\rho(x) \notin \bar{\rho}(\varphi)$;
- $\bar{\rho}(\varphi_1 \subseteq \varphi_2) = M$ iff $\bar{\rho}(\varphi_1) \subseteq \bar{\rho}(\varphi_2)$; and $\bar{\rho}(\varphi_1 \subseteq \varphi_2) = \emptyset$ iff $\bar{\rho}(\varphi_1) \not\subseteq \bar{\rho}(\varphi_2)$.

Proof. These can be proved by simply applying the definitions.

(Case $\lceil\varphi\rceil$): $\bar{\rho}(\lceil\varphi\rceil) = M$ iff $\lceil\_\rceil_M \cdot \bar{\rho}(\varphi) = M$ iff there exists $a \in \bar{\rho}(\varphi)$ iff $\bar{\rho}(\varphi) \neq \emptyset$. Otherwise, $\bar{\rho}(\lceil\varphi\rceil) = \emptyset$ iff $\lceil\_\rceil_M \cdot \bar{\rho}(\varphi) = \emptyset$ iff there exists no $a \in \bar{\rho}(\varphi)$ iff $\bar{\rho}(\varphi) = \emptyset$.

(Case $\lfloor\varphi\rfloor$): $\bar{\rho}(\lfloor\varphi\rfloor) = M$ iff $\bar{\rho}(\neg\lceil\neg\varphi\rceil) = M$ iff $\bar{\rho}(\lceil\neg\varphi\rceil) = \emptyset$ iff $\bar{\rho}(\neg\varphi) = \emptyset$ iff $\bar{\rho}(\varphi) = M$. Otherwise, $\bar{\rho}(\lfloor\varphi\rfloor) = \emptyset$ iff $\bar{\rho}(\neg\lceil\neg\varphi\rceil) = \emptyset$ iff $\bar{\rho}(\lceil\neg\varphi\rceil) = M$ iff $\bar{\rho}(\neg\varphi) = M$ iff $\bar{\rho}(\varphi) = \emptyset$.

(Case $\varphi_1 = \varphi_2$): $\bar{\rho}(\varphi_1 = \varphi_2) = M$ iff $\bar{\rho}(\lfloor\varphi_1 \leftrightarrow \varphi_2\rfloor) = M$ iff $\bar{\rho}(\varphi_1 \leftrightarrow \varphi_2) = M$ iff $\bar{\rho}(\varphi_1) \vartriangle \bar{\rho}(\varphi_2) = \emptyset$ iff $\bar{\rho}(\varphi_1) = \bar{\rho}(\varphi_2)$. Otherwise, $\bar{\rho}(\varphi_1 = \varphi_2) = \emptyset$ iff $\bar{\rho}(\lfloor\varphi_1 \leftrightarrow \varphi_2\rfloor) = \emptyset$ iff $\bar{\rho}(\varphi_1 \leftrightarrow \varphi_2) = \emptyset$ iff $\bar{\rho}(\varphi_1) \vartriangle \bar{\rho}(\varphi_2) = M$ iff $\bar{\rho}(\varphi_1) \neq \bar{\rho}(\varphi_2)$.

(Case $x \in \varphi$): $\bar{\rho}(x \in \varphi) = M$ iff $\bar{\rho}(\lceil x \wedge \varphi\rceil) = M$ iff $\bar{\rho}(x \wedge \varphi) \neq \emptyset$ iff $\{\rho(x)\} \cap \bar{\rho}(\varphi) \neq \emptyset$ iff $\rho(x) \in \bar{\rho}(\varphi)$. Otherwise, $\bar{\rho}(x \in \varphi) = \emptyset$ iff $\bar{\rho}(\lceil x \wedge \varphi\rceil) = \emptyset$ iff $\bar{\rho}(x \wedge \varphi) = \emptyset$ iff $\{\rho(x)\} \cap \bar{\rho}(\varphi) = \emptyset$ iff $\rho(x) \notin \bar{\rho}(\varphi)$.

(Case $\varphi_1 \subseteq \varphi_2$): $\bar{\rho}(\varphi_1 \subseteq \varphi_2) = M$ iff $\bar{\rho}(\lfloor\varphi_1 \to \varphi_2\rfloor) = M$ iff $\bar{\rho}(\varphi_1 \to \varphi_2) = M$ iff $M \setminus (\bar{\rho}(\varphi_1) \setminus \bar{\rho}(\varphi_2)) = M$ iff $\bar{\rho}(\varphi_1) \setminus \bar{\rho}(\varphi_2) = \emptyset$ iff $\bar{\rho}(\varphi_1) \subseteq \bar{\rho}(\varphi_2)$. Otherwise, $\bar{\rho}(\varphi_1 \subseteq \varphi_2) = \emptyset$ iff $\bar{\rho}(\lfloor\varphi_1 \to \varphi_2\rfloor) = \emptyset$ iff $\bar{\rho}(\varphi_1 \to \varphi_2) \neq M$ iff $M \setminus (\bar{\rho}(\varphi_1) \setminus \bar{\rho}(\varphi_2)) \neq M$ iff $\bar{\rho}(\varphi_1) \setminus \bar{\rho}(\varphi_2) \neq \emptyset$ iff $\bar{\rho}(\varphi_1) \not\subseteq \bar{\rho}(\varphi_2)$.

□

C Proofs of Results in Section 3

Proposition 7. For theories Γ with definedness, we have

- $\Gamma \vdash \varphi = \varphi$
- $\Gamma \vdash \varphi_1 = \varphi_2$ and $\Gamma \vdash \varphi_2 = \varphi_3$ implies $\Gamma \vdash \varphi_1 = \varphi_3$
- $\Gamma \vdash \varphi_1 = \varphi_2$ implies $\Gamma \vdash \varphi_2 = \varphi_1$
- $\Gamma \vdash \varphi_1 = \varphi_2$ implies $\Gamma \vdash \psi[\varphi_1/x] = \psi[\varphi_2/x]$

Here ψ can be any pattern, not only application context. In other words, the *indiscernibility of identicals* holds in AML.

Proof. See [10, Lemma 50 and 60].

□

Theorem 8 (Soundness Theorem)**.** $\Gamma \vdash \varphi$ *implies* $\Gamma \vDash \varphi$.

Proof. See [10, Theorem 13].

□

D Some Discussion about Definedness Symbol

Note the following property about definedness.

Proposition 21. Let M be an AML model satisfying (Definedness). Then $M \cdot a = M$ for all $a \in M$, i.e., for all $a, b \in M$ there exists $c \in M$ such that $b \in c \cdot a$.

Proof. Let $\lceil \_ \rceil_M \subseteq M$ be the interpretation of definedness in M. Since M satisfies (Definedness), we have $\lceil \_ \rceil_M \cdot a = M$ for all $a \in M$. Then by pointwise extension, $M \cdot a = M$.

□

For convenience and simplicity, when we construct an AML model, we assume there is a special element, say $\$ \in M$, such that $\$ \cdot a = M$ for all $a \in M$. Then, we use $\$ as the interpretation of definedness in our model.

E Proof of Results in Section 4

Proposition 10. $\vdash \forall x{:}s.\, \varphi = \neg \exists x{:}s.\, \neg \varphi$

Proof. We prove the first one as follows:

$$\begin{aligned}
\forall x{:}s.\, \varphi &= \forall x.\, x \in [\![s]\!] \to \varphi && \text{// by definition} \\
&= \neg \exists x.\, \neg (x \in [\![s]\!] \to \varphi) && \text{// by FOL reasoning} \\
&= \neg \exists x.\, x \in [\![s]\!] \wedge \neg \varphi && \text{// by FOL reasoning} \\
&= \neg \exists x{:}s.\, \varphi && \text{// by definition}
\end{aligned}$$

The second can be proved in the same way.

□

Proposition 12. For MSFOL terms t_s and formulas φ,

$$\Gamma^{\mathsf{MSFOL}} \vdash \mathrm{ws}(t_s) \to \exists y{:}s.\, t_s = y$$
$$\Gamma^{\mathsf{MSFOL}} \vdash \mathrm{ws}(\varphi) \to (\varphi = \top) \vee (\varphi = \bot).$$

Proof. We first prove $\Gamma^{\mathsf{MSFOL}} \vdash \mathrm{ws}(t) \to \exists y{:}s.\, t = s$. We carry out structural induction on t. When t is a variable $x{:}s$, the proof obligation becomes $\mathrm{ws}(x{:}s) \to \exists y{:}s.\, x = y$, i.e., $x \in [\![s]\!] \to \exists y.\, y \in [\![s]\!] \wedge x = y$, which trivially holds. When t is a term $f(t_1, \ldots, t_n)$ with $f \in F_{s_1 \ldots s_n, s}$ and t_1 has sort $s_1, \ldots, t_n$ has sort s_n, the proof obligation becomes $\mathrm{ws}(f(t_1, \ldots, t_n)) \to \exists y{:}s.\, f(t_1, \ldots, t_n) = t$. Recall that MSFOL term $f(t_1, \ldots, t_n)$ is translated to AML pattern $f\, t_1 \cdots t_n$, the proof obligation becomes $\mathrm{ws}(f\, t_1 \cdots t_n) \to \exists y{:}s.\, f\, t_1 \cdots t_n = t$. By definition of the predicate ws, we have that $\Gamma^{\mathsf{MSFOL}} \vdash \mathrm{ws}(f\, t_1 \cdots t_n) \leftrightarrow \mathrm{ws}(t_1) \wedge \cdots \wedge \mathrm{ws}(t_n)$, because $\mathrm{FV}(f\, t_1 \cdots t_n) = \bigcup_i \mathrm{FV}(t_i)$. Additionally, by induction hypothesis on $t_1, \ldots, t_n$, we have that $\Gamma^{\mathsf{MSFOL}} \vdash \mathrm{ws}(t_i) \to \exists y_i{:}s_i.\, t_i = y_i$, for $1 \leq i \leq n$. Putting them together, we only need to prove $\Gamma^{\mathsf{MSFOL}} \vdash \bigwedge_i \exists y_i{:}s_i.\, t_i = y_i \to \exists y{:}s.\, f\, t_1 \cdots t_n = y$. By FOL and equational reasoning, we only need to prove $\Gamma^{\mathsf{MSFOL}} \vdash \forall y_1{:}s_1 \ldots \forall y_n{:}s_n.\, \exists y{:}s.\, f\, t_1 \cdots t_n = y$, which is exactly axiom (Function) for f.

Next, we prove $\Gamma^{\mathsf{MSFOL}} \vdash \mathrm{ws}(\varphi) \to (\varphi = \top) \vee (\varphi = \bot)$. We carry out structural induction on φ. Note that φ is an MSFOL formula, i.e., an AML pattern built from $\bot$, or $\varphi_1 \to \varphi_2$, or $\exists x{:}s.\, \varphi$, or $p(t_1, \ldots, t_n)$ for predicate $p \in \Pi_{s_1 \ldots s_n}$. Since the predicate-ness of patterns propagate through all AML logic connectives and constructs, we only need to prove the two base cases: when φ is $\bot$ and when φ is $p(t_1, \ldots, t_n)$. For the former, it is trivial that the conclusion holds. For the latter, it is a direct consequence of axiom (Predicate) for p.

□

Theorem 13. $\Omega \vdash_{\mathsf{MSFOL}} \varphi$ *implies* $\Gamma^{\mathsf{MSFOL}} \vdash \varphi^{\mathsf{MSFOL}}$.

Proof. We carry out structural induction on the length of Hilbert-style proof $\Omega \vdash_{\mathsf{MSFOL}} \varphi$. Note that MmL and AML have a lot of proof rules in common. We only need to prove that the following two MSFOL specific proof rules are derivable by the AML proof system:

$$\text{(Sorted } \exists\text{-Quantifier)} \quad \varphi[t_s/x{:}s] \to \exists x{:}s.\, \varphi$$

$$\text{(Sorted } \exists\text{-Generalization)} \quad \frac{\varphi_1 \to \varphi_2}{(\exists x{:}s.\, \varphi_1) \to \varphi_2} \text{ if } x{:}s \notin \mathrm{FV}(\varphi_2)$$

For (Sorted $\exists$-Quantifier), we need to prove that $\Gamma^{\mathsf{MSFOL}} \vdash \varphi[t_s/x{:}s] \to \exists x.\, x \in [\![s]\!] \wedge \varphi$. To prove that, we just need to prove t_s is a functional pattern of sort s in AML. This has been proved in Proposition 12.

For (Sorted $\exists$-Generalization), we only need to apply AML's unsorted ($\exists$-Generalization), and the fact that $\Gamma^{\mathsf{MSFOL}} \vdash (\exists x{:}s.\, \varphi_1) \to (\exists x.\, \varphi_1)$. The latter holds by standard FOL reasoning.

To sum up, we have proved the conclusion by induction on the length of the Hilbert-style proof $\Omega \vdash_{\mathsf{MSFOL}} \varphi$ that Γ^{MSFOL} preserves all MSFOL reasoning.

□

The following proposition is useful in reasoning about the semantics of sorted quantification.

Proposition 22. Let M be a model, s be a sort, whose domain is denoted/defined as $M_s = [\![s_M]\!]_M$. Then for any valuation ρ, $\bar{\rho}(\exists x{:}s.\, \varphi) = \bigcup_{a \in M_s} \overline{\rho[a/x]}(\varphi)$ and $\bar{\rho}(\forall x{:}s.\, \varphi) = \bigcap_{a \in M_s} \overline{\rho[a/x]}(\varphi)$

Proof. We prove for $\exists x{:}s.\, \varphi$. The proof of the other one is similar. By definition, $\bar{\rho}(\exists x{:}s.\, \varphi) = \bar{\rho}(\exists x.\, x \in [\![s]\!] \wedge \varphi) = \bigcup_{a \in M} \left(\overline{\rho[a/x]}(x \in [\![s]\!]) \cap \overline{\rho[a/x]}(\varphi)\right)$. Note that $\overline{\rho[a/x]}(x \in [\![s]\!]) = M$ iff $\rho[a/x](x) \in \overline{\rho[a/x]}([\![s]\!])$, i.e., $a \in M_s$; and $\overline{\rho[a/x]}(x \in [\![s]\!]) = \emptyset$ iff $a \notin M_s$; see Proposition 4.

Therefore, $\bar{\rho}(\exists x{:}s.\, \varphi) = \bigcup_{a \in M_s} \left(\overline{\rho[a/x]}(x \in [\![s]\!]) \cap \overline{\rho[a/x]}(\varphi)\right) = \bigcup_{a \in M_s} \overline{\rho[a/x]}(\varphi)$. □

Theorem 15. *For MSFOL theory Ω and model $A \models_{\mathsf{MSFOL}} \Omega$, there is an AML model $M \models \Gamma^{\mathsf{MSFOL}}$ whose MSFOL-restricted model is exactly A. In addition, $\Gamma^{\mathsf{MSFOL}} \models \varphi^{\mathsf{MSFOL}}$ iff $\Omega \models_{\mathsf{MSFOL}} \varphi$.*

Proof. Assume an MSFOL model $A = (\{A_s\}_{s \in S}, \{f_A\}_{f \in F}, \{p_A\}_{p \in \Pi})$. We give an explicit construction on M. Firstly, we define its domain, which (by abuse of notation) is also denoted as M. We let M contain/include all the following elements/sets:

- \$, which is a distinguished element denoting definedness; see Appendix D;
- #, which is a distinguished element denoting the domain symbol;
- S, the sort set;
- A_s, for all $s \in S$;
- $[A_{s_i} \to [A_{s_{i+1}} \to [\cdots \to [A_{s_n} \to A_s] \dots]]]$ for all $F_{s_1 \dots s_n, s} \neq \emptyset$ and $1 \leq i \leq n$. Here $[A \to B]$ denotes the set of all functions from A to B; intuitively, it is used to define the interpretations of functions and their partial applications;
- $A_{s_i} \times \cdots \times A_{s_n}$ for all $\Pi_{s_1 \dots s_n} \neq \emptyset$ and $1 \leq i \leq n$; intuitively, it is used to define the interpretations of predicates and their partial applications.

Next, we define the following interpretations of constants:

- $\lceil \_ \rceil_M = \{\$\}$ and $[\![\_]\!]_M = \{\#\}$;
- $s_M = \{s\}$ for all $s \in S$;
- $f_M = \{f_A\}$ for all $f \in F_{s_1 \dots s_n, s}$; note that $f_A \colon A_{s_1} \times \cdots \times A_{s_n} \to A_s$, under the *curry-uncurry isomorphism*, is an element in $[A_{s_1} \to [A_{s_2} \to [\cdots \to [A_{s_n} \to A_s] \cdots]]]$. Therefore, $f_A \in M$ and that f_M is well-defined.
- for $p \in \Pi_\epsilon$, i.e., constant predicate (atomic proposition) whose truth value is determined by the structure A, we define $p_M = \emptyset$ whenever p_A is always false and $p_M = M$ whenever p_A is always true;
- for $p \in \Pi_{s_1 \dots s_n}$ for $n \geq 1$, we define $p_M = p_A$; note that $p_A \subseteq A_{s_1} \times \cdots \times A_{s_n}$ is a subset of M, so p_M is well-defined.

Next, we define the application in M as the following:

- $\$ \bullet a = M$ for all $a \in M$;
- $\# \bullet s = A_s$ for all $s \in S$; note that $[\![s_M]\!]_M = \# \bullet \{s\} = A_s$;
- $f \bullet a = \{f(a)\}$ for all $f \in [A \to B]$ and $a \in A$; that is, application is interpreted as the normal function application, if the first argument is a function and the second is an element of the appropriate sort;
- for $p \subseteq A_s$ for some $s \in S$, we define $p \bullet a = M$ iff $a \in p$ and $p \bullet a = \emptyset$ iff $a \not\in p$; intuitively, p is the interpretation of a unary symbol, so when it is applied to an argument, the result is either $\emptyset$ (indicating false) or M (indicating true);
- $p \bullet a = \{b \mid (a, b) \in p\}$ for all $p \subseteq A \times B$ and $a \in A$; that is, application is interpreted as projections if the first argument is an n-ary relation for $n \geq 2$ and the second is an element of the appropriate sort.

Next, we verify that M satisfies all axioms in $\Gamma^{\mathsf{MSFOL}} \setminus \Omega^{\mathsf{MSFOL}}$; for axioms in Ω^{MSFOL}, we will prove that M satisfies them later. (Definedness) is satisfied by definition. (Nonempty Sort) is satisfied because $A_s \neq \emptyset$ for all $s \in S$. (Function) for every $f \in F_{s_1 \dots s_n, s}$ is satisfied because f_M is defined by f_A and the application $\_\bullet\_$ is interpreted as the normal function application. (Predicate) for every $p \in \Pi_{s_1 \dots s_n}$ is satisfied because p_M is defined as a relation and the application $\_\bullet\_$ is interpreted as the projection of relations, until it reaches the last argument (i.e., for unary relations), in which case either $\emptyset$ or M is returned. Therefore, $M \models \Gamma^{\mathsf{MSFOL}} \setminus \Omega^{\mathsf{MSFOL}}$.

Next, we prove the MSFOL-restricted model of M, denoted B, is exactly A. By definition, $B = (\{B_s\}_{s \in S}, \{f_B\}_{f \in F})$ is an MSFOL structure consisting of:

- the domain $B_s = [\![s_M]\!]_M$, which equals A_s as we showed above, for all $s \in S$;
- the interpretation f_B, for $f \in F_{s_1 \dots s_n, s}$, is defined such that $\{f_B(a_1, \dots, a_n)\} = f_M a_1 \cdots a_n$ for all $a_1 \in A_{s_1}, \dots, a_n \in A_{s_n}$; note that $f_M a_1 \cdots a_n = \{f_A\} a_1 \cdots a_n = \{f_A(a_1, \dots, a_n)\}$, so we have $f_B(a_1, \dots, a_n) = f_A(a_1, \dots, a_n)$, for all $a_1 \in A_{s_1}, \dots, a_n \in A_{s_n}$.

Therefore, B is exactly A (up to isomorphism).

Finally, we show that for any MSFOL formula φ, $A \models_{\mathsf{MSFOL}} \varphi$ iff $M \vDash \varphi^{\mathsf{MSFOL}}$. This immediately implies that $M \vDash \Omega^{\mathsf{MSFOL}}$ and that $\Omega \models_{\mathsf{MSFOL}} \varphi$ iff $\Gamma^{\mathsf{MSFOL}} \vDash \varphi^{\mathsf{MSFOL}}$. For clarity, we move rest of the proof to Lemma 23. □

Lemma 23. *Under the notations of the proof of Theorem 15, we have that every MSFOL valuation derives a corresponding AML valuation ρ_v, with $\rho_v(x) = v(x{:}s)$ for all MSFOL variable x. Then for every MSFOL term t of sort s and MSFOL formula φ, we have $\bar{v}(t) = \overline{\rho_v}(t)$ and that (a) $\bar{v}(\varphi) = \mathit{true}$ iff $\overline{\rho_v}(\varphi^{\mathsf{MSFOL}}) = M$, and (b) $\bar{v}(\varphi) = \mathit{false}$ iff $\overline{\rho_v}(\varphi^{\mathsf{MSFOL}}) = \emptyset$.*

Proof. We fist prove $\bar{v}(t) = \overline{\rho_v}(t)$ by structural induction on t. When t is $x{:}s$, the conclusion holds by definition of ρ_v. When t is $f(t_1, \dots, t_n)$, the conclusion holds by the induction hypotheses for $t_1, \dots, t_n$ and the definition of f_M.

Next we prove both (a) and (b) simultaneously by carrying out structural induction on φ. When φ is $\bot$, we have $\bar{v}(\varphi) = \mathit{false}$, always, and $\overline{\rho_v}(\varphi^{\mathsf{MSFOL}}) = \overline{\rho_v}(\mathrm{ws}(\bot) \to \bot = \top) = \emptyset$. When φ is $\varphi_1 \wedge \varphi_2$, we have $\bar{v}(\varphi) = \bar{v}(\varphi_1 \wedge \varphi_2)$. Suppose it is true, then $\bar{v}(\varphi_1) = \bar{v}(\varphi_2) = \mathit{true}$ by the semantics of MSFOL. By induction hypotheses on φ_1 and φ_2, we have $\overline{\rho_v}(\varphi_1^{\mathsf{MSFOL}}) = \overline{\rho_v}(\varphi_2^{\mathsf{MSFOL}}) = M$. Therefore, $\overline{\rho_v}(\varphi^{\mathsf{MSFOL}}) = \overline{\rho_v}(\mathrm{ws}(\varphi) \to \varphi = \top) = \overline{\rho_v}((\mathrm{ws}(\varphi_1) \to \varphi_1 = \top) \wedge (\mathrm{ws}(\varphi_2) \to \varphi_2 = \top) = M$. We can prove the case when $\bar{v}(\varphi_1 \wedge \varphi_2) = \mathit{false}$ similarly and both reasonings are reversible. When φ is $\exists x{:}s.\varphi_1$, we have $\bar{v}(\varphi) = \bar{v}(\exists x{:}s.\varphi_1)$. Suppose it is true, then there exists an element $a \in A_s$ such that $\overline{v[a/x{:}s]}(\varphi_1) = \mathit{true}$. By inductive hypothesis, $\overline{\rho_v[a/x]}(\varphi_1^{\mathsf{MSFOL}}) = M$, i.e., $\overline{\rho_v[a/x]}(\mathrm{ws}(\varphi) \to \varphi_1 = \bot) = M$. Note that the new valuation $\rho_v[a/x]$ satisfies $\mathrm{ws}(\varphi_1)$, so we have $\overline{\rho_v[a/x]}(\varphi_1 = \bot) = M$, i.e., $\overline{\rho_v[a/x]}(\varphi_1) = M$, i.e., $\bar{\rho}(\exists x{:}s.\varphi_1) = M$, i.e., $\bar{\rho}(\mathrm{ws}(\varphi) \to \exists x{:}s.\varphi_1 = \top)$, i.e., $\bar{\rho}(\varphi^{\mathsf{MSFOL}}) = M$. Next, suppose $\bar{v}(\varphi) = \bar{v}(\exists x{:}s.\varphi_1) = \mathit{false}$. Follow a similar reasoning we show that $\overline{\rho_v[a/x]}(\mathrm{ws}(\varphi_1) \to \varphi_1 = \top) = \emptyset$ for all $a \in A_s$, i.e., $\bar{\rho}(\mathrm{ws}(\varphi) \to \exists x{:}s.\varphi_1 = \top) = \emptyset$, which finishes the proof. Finally, when φ is $p(t_1, \dots, t_n)$, the conclusion follows by that $\bar{v}(t) = \overline{\rho_v}(t)$ and the definition of p_M.

□

F Instance: Constructors and Term Algebras

Constructors are extensively used in building programs and data, as well as semantic structures to define and reason about languages and programs. They generate *term algebras*, whose elements are terms and functions are constructors that build terms. In the broader context of defining formal semantics of languages, term algebras, as a special case of *initial algebras*, play an important role and have led to many applications and tools (see, e.g., [27, 28, 12]). In this section, we show how to define constructors and term algebras using the least fixpoint patterns that support *inductive reasoning*. The technique shown here can be naturally applied to defining *initial algebra semantics* [24, 27], which is an important theoretical framework whose applications include context free grammars for formal syntax and denotational semantics for formal semantics.

Let us fix an MSFOL signature $(\{\mathit{Term}\}, C, \emptyset)$ with one sort *Term*, a set C of functions, called *constructors*, where at least one of them is a constant, and no predicates. The same technique applies to multiple sorts. We use $c, d, \dots$ to denote constructors. The set of *C-terms*, denoted as T_C, is defined as follows:

$$\begin{aligned} t ::= {} & c \in C_{\epsilon, \mathit{Term}} \\ & \mid c(t_1, \dots, t_n) \text{ for } c \in C_{\mathit{Term} \dots \mathit{Term}, \mathit{Term}} \text{ with } n \geq 1 \end{aligned}$$

The *C-term algebra* is an MSFOL structure $T = (T_C, \{c_T\}_{c \in C}, \emptyset)$ with domain T_C and interpretations $c_T \colon T_C \times \cdots \times T_C \to T_C$ for $c \in C_{Term \ldots Term, Term}$ such that $c_T(t_1, \ldots, t_n)$ is $c(t_1, \ldots, t_n)$ when $n \geq 1$ and c when $n = 0$.

Now we define an AML theory Γ^{Term} that faithfully captures the term algebra T. Since T is an MSFOL structure, we let Γ^{Term} contain all axioms in Section 4. In addition, we specify (1) all constructors are injective functions; and (2) the domain of *Term* is precisely T_C, as the following axioms:

$$
\begin{aligned}
&\text{(No Confusion I, for all dinstinct } c, d \in C \text{ of arities } n \text{ and } m, \text{)} \\
&\quad \forall x_1 : Term \cdots \forall x_n : Term \,.\, \forall y_1 : Term \cdots \forall y_m : Term \,. \\
&\qquad c x_1 \cdots x_n \neq d y_1 \cdots y_m \\
&\text{(No Confusion II, for all } c \in C\text{)} \\
&\quad \forall x_1 : Term \cdots \forall x_n : Term \,.\, \forall x'_1 : Term \cdots \forall x'_n : Term \,. \\
&\qquad\qquad c x_1 \cdots x_n = c x'_1 \cdots x'_n \to x_1 = x'_1 \wedge \cdots \wedge x_n = x'_n \\
&\text{(Inductive Domain)} \quad [\![Term]\!] = \mu D \,.\, \bigvee_{c \in C} \underbrace{c\, D \cdots D}_{\text{as many } D\text{'s as the arity of } c}
\end{aligned}
$$

Intuitively, (No Confusion I) says different constructors build different terms; (No Confusion II) says that constructors are injective functions; (Inductive Domain) forces that $[\![Term]\!]$ is the smallest set closed under all constructors, yielding exactly T_C, as shown in the following proposition.

Proposition 24. For any AML model $M \models \Gamma^{\mathsf{Term}}$, its MSFOL-restricted model as defined in Definition 14 is exactly T. In particular, $[\![Term]\!]_M = T_C$.

The proof follows from the proof of [9, Proposition 22], which there was done for the more general context of MmL, see Section G. Therefore, AML can define term algebras *up-to-isomorphism*, and not only up-to-elementary-equivalence as with FOL [41].

Proof. By Theorem 30 and [9, Proposition 22]. □

As a direct corollary of Proposition 24, the AML axiomatization Γ^{NAT} of $(\mathbb{N}, +, \times)$ as defined in Section 4.5 captures the standard model of natural numbers, which is a term algebra generated from $\{0, succ\}$, up to isomorphism. This is shown as the following proposition.

Proposition 18. The MSFOL-restricted model M of Γ^{NAT} is exactly $(\mathbb{N}, +, \times)$, up to isomorphism.

Proof. By applying Proposition 24 on Γ^{NAT}. □

Proposition 17. $\Gamma^{\mathsf{NAT}} \vdash 0 \in X \wedge (\forall y : Nat \,.\, y \in X \to (succ\ y) \in X) \to \forall x : Nat \,.\, x \in X$.

Proof. To prove $\forall x : Nat \,.\, x \in X$, we just need to show that $[\![Nat]\!] \subseteq X$. Recall that $[\![Nat]\!] = \mu D \,.\, 0 \vee succ\ D$, so we need to show $\mu D \,.\, 0 \vee succ\ D \to X$. By (Knaster-Tarski), it suffices to prove $0 \vee succ\ X \to X$, which boils down to proving (1) $0 \to X$ and (2) $succ\ X \to X$. Firstly, (1) is directly proved by $0 \in X$. Secondly, (2) is proved by proving $z \in succ\ X \to z \in X$ for all z, which then boils down to proving $\exists y \,.\, y \in X \wedge z = succ\ y \to z \in X$, which is then proved from $\forall y : Nat \,.\, y \in X \to succ\ y \in X$. □

G Instance: Matching μ-Logic (MmL)

Here we show in full detail how to define matching μ-logic (abbreviated MmL) [9]. MmL is a many-sorted FOL variant for *pattern matching*, with direct support for inductive reasoning and least fixpoints. We choose MmL because it is so far the best unifying logic as the foundation of the $\mathbb{K}$ framework and many important logics such as FOL with least fixpoints, separation logic, modal logic, temporal logics, dynamic logic, and reachability logic have all been defined as theories and

notations in MmL. The fact that we can define MmL in AML immediately shows that AML can also subsume the aforementioned logics.

As we will see, MmL is essentially an extension of AML with many-sorted syntax and structures or that AML is the fragment of MmL with only one sort, one binary symbol (for application), and several constant symbols. The results shown in this section show that we can safely eliminate the many-sorted infrastructure of MmL without losing any generality or power in model specification or logic reasoning.

G.1 Matching μ-Logic Preliminaries

Definition 25 ([9, Definition 19])**.** A *matching μ-logic signature* (S, V, Σ) contains a sort set S, a variable set $V = EV \cup SV$ with *element variables* $EV = \{EV_s\}_{s\in S}$ and *set variables* $SV = \{SV_s\}_{s\in S}$, and a many-sorted symbol set $\Sigma = \{\Sigma_{s_1\dots s_n,s}\}_{s_1,\dots,s_n,s\in S}$. MmL formulas, also called *patterns*, are:

$$\begin{aligned}\varphi_s ::= {} & x{:}s \in EV_s \mid X{:}s \in SV_s \mid \sigma(\varphi_{s_1},\dots,\varphi_{s_n}) \text{ where } \sigma \in \Sigma_{s_1\dots s_n,s}\\ & \mid \varphi_s \wedge \varphi'_s \mid \neg\varphi_s \mid \exists x{:}s'.\varphi_s \mid \mu X{:}s.\varphi_s \text{ where } \varphi_s \text{ positive in } X{:}s\end{aligned}$$

An MmL structure $M = (\{M_s\}_{s\in S}, \{\sigma_M\}_{\sigma\in\Sigma})$ contains a nonempty domain M_s for every $s \in S$ and an interpretation $\sigma_M\colon M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$ for every $\sigma \in \Sigma_{s_1\dots s_n,s}$. We extend σ_M to its *pointwise extension*, $\sigma_M\colon \mathcal{P}(M_{s_1}) \times \cdots \times \mathcal{P}(M_{s_n}) \to \mathcal{P}(M_s)$, defined as $\sigma_M(A_1,\dots,A_n) = \bigcup_{a_i\in A_i, 1\le i\le n} \sigma(a_1,\dots,a_n)$ for $A_i \subseteq M_{s_i}$, $1 \le i \le n$. An M-valuation $\rho\colon V \to M \cup \mathcal{P}(M)$ is one such that $\rho(x{:}s) \in M_s$ and $\rho(X{:}s) \subseteq M_s$ for all $x{:}s, X{:}s \in V$. Its *extension* $\bar\rho$ is defined:

$$\begin{aligned}&\bar\rho(x{:}s) = \{\rho(x{:}s)\} \quad \bar\rho(X{:}s) = \rho(X{:}s) \quad \bar\rho(\varphi_s \wedge \varphi'_s) = \bar\rho(\varphi_s) \cap \bar\rho(\varphi'_s)\\ &\bar\rho(\sigma(\varphi_{s_1},\dots,\varphi_{s_n})) = \sigma_M(\bar\rho(\varphi_{s_1}),\dots,\bar\rho(\varphi_{s_n})) \text{ for } \sigma \in \Sigma_{s_1\dots s_n,s}\\ &\bar\rho(\neg\varphi_s) = M \setminus \bar\rho(\varphi_s) \quad \bar\rho(\exists x{:}s'.\varphi_s) = \textstyle\bigcup_{a\in M_{s'}} \overline{\rho[a/x{:}s']}(\varphi_s)\\ &\bar\rho(\mu X{:}s.\varphi_s) = \mu\mathcal{F}^{\rho}_{\varphi,X:s} \text{ with } \mathcal{F}^{\rho}_{\varphi,X:s}(A) = \overline{\rho[A/X{:}s]}(\varphi) \text{ for } A \subseteq M_s\end{aligned}$$

We write $M \vDash_{\mathsf{MmL}} \varphi_s$ iff $\bar\rho(\varphi) = M_s$ for all ρ, and if Ω is an MmL theory then we write $\Omega \vDash_{\mathsf{MmL}} \varphi_s$ iff $M \vDash_{\mathsf{MmL}} \varphi_s$ for all MmL models $M \vDash_{\mathsf{MmL}} \Omega$. Provability relation $\Omega \vdash_{\mathsf{MmL}} \varphi_s$ is defined in [9, Fig. 1].

G.2 Defining MmL Syntax and Proofs in AML

Given an MmL signature (S, V, Σ), we define a corresponding AML signature Σ^{MmL} and an AML theory Σ^{MmL} that captures (S, V, Σ). Also, we show how to translate an MmL theory Ω to an AML theory Ω^{VALID}, preserving the semantics and provability. We basically follow the definition of MSFOL shown in Section 4, with some special treatments to deal with MmL symbols and connectives.

Let us define $\mathit{Sort} \in \Sigma^{\mathsf{MmL}}$ to represent the sort set and $s \in \Sigma^{\mathsf{MmL}}$ to represent each sort $s \in S$, as in MSFOL, with axiom $[\![s]\!] \neq \bot$ that specifies the nonemptiness of domains. For every $\sigma \in \Sigma_{s_1\dots s_n,s}$, we define a corresponding constant $\sigma \in \Sigma^{\mathsf{MmL}}$ and use the following axiom to specify its arity:

$$(\textsc{Arity}) \quad \sigma\,[\![s_1]\!] \cdots [\![s_n]\!] \subseteq [\![s]\!]$$

Intuitively, (Arity) says the result of applying σ to arguments of appropriate sorts is contained in the domain of s. Note that (Arity) is weaker than (Function) in Section 4, as the latter requires additionally that σ produces singletons.

Let Γ^{MmL} denotes the set of axioms we have defined so far. We translate an MmL pattern φ_s to an AML pattern $\varphi_s^{\mathsf{MmL2AML}}$ as follows:

$$\begin{aligned}&(x{:}s)^{\mathsf{MmL2AML}} \equiv x \quad (X{:}s)^{\mathsf{MmL2AML}} \equiv X \quad \exists x{:}s'.\varphi_s \equiv \exists x{:}s'.\varphi_s^{\mathsf{MmL2AML}}\\ &(\varphi_s \wedge \psi_s)^{\mathsf{MmL2AML}} \equiv \varphi_s^{\mathsf{MmL2AML}} \wedge \psi_s{}^{\mathsf{MmL2AML}} \quad \mu X{:}s.\varphi_s \equiv \mu X.\varphi_s^{\mathsf{MmL2AML}}\\ &(\sigma(\varphi_{s_1},\dots,\varphi_{s_n}))^{\mathsf{MmL2AML}} \equiv \sigma\,\varphi_{s_1}^{\mathsf{MmL2AML}} \dots \varphi_{s_n}^{\mathsf{MmL2AML}}\\ &(\neg\varphi_s)^{\mathsf{MmL2AML}} \equiv \neg_s\varphi_s^{\mathsf{MmL2AML}} \text{ where } \neg_s\varphi_s^{\mathsf{AML}} \equiv \neg\varphi_s^{\mathsf{AML}} \wedge [\![s]\!] \text{ called } \textit{sorted negation}\end{aligned}$$

In short, the MmL2AML translation is almost an identity function. The arities of MmL symbols have been specified in Γ^{MmL}. The sorts of MmL variables are dropped and will be recovered in Proposition 26. by the "well-sorted" patterns $\mathrm{ws}(\varphi)$ defined as in Definition 11. One special treatment has been made to faithfully capture MmL negation $\neg\varphi_s$, which is matched by all elements *of sort s* that does not match φ_s. Therefore, we define AML *sorted negation* $\neg_s\varphi \equiv (\neg\varphi) \wedge [\![s]\!]$ to match the semantics of MmL negation.

The following proposition shows that MmL patterns are wellformed AML patterns if sort constraints on variables are met. Note that we need to extend the definition of "well-sorted" patterns $\mathrm{ws}(\varphi)$ defined in Definition 11. There, it was defined for only MSFOL variables, which are like AML element variables. For AML set variables, we define the well-sorted pattern/predicate for X as $X \subseteq [\![s]\!]$, that is, we change $\in$ to $\subseteq$, because X can be assigned to any set, rather than only a singleton.

Proposition 26. For MmL pattern φ_s of sort s, $\Gamma^{\mathsf{MmL}} \vdash \mathrm{ws}(\varphi_s) \to \varphi_s^{\mathsf{AML}} \subseteq [\![s]\!]$.

Proof. We carry out structural induction on φ_s. When φ_s is $x{:}s$, we need to prove $\Gamma^{\mathsf{MmL}} \vdash \mathrm{ws}(x{:}s) \to x \subseteq [\![s]\!]$, which holds trivially. When φ_s is $X{:}s$, we need to prove $\Gamma^{\mathsf{MmL}} \vdash \mathrm{ws}(X{:}s) \to X \subseteq [\![s]\!]$, which holds trivially. When φ_s is $\varphi_1 \wedge \varphi_2$, we need to prove, by induction hypothesis, that $\Gamma^{\mathsf{MmL}} \vdash \varphi_1^{\mathsf{AML}} \subseteq [\![s]\!] \wedge \varphi_2^{\mathsf{AML}} \subseteq [\![s]\!] \to \varphi_s^{\mathsf{AML}} \subseteq [\![s]\!]$, which holds trivially. When φ_s is $\neg\varphi_1$, we nee to prove, by induction hypothesis, that $\Gamma^{\mathsf{MmL}} \vdash \varphi_1^{\mathsf{AML}} \subseteq [\![s]\!] \to (\neg\varphi_1^{\mathsf{AML}} \wedge [\![s]\!]) \subseteq [\![s]\!]$, which holds by simple FOL reasoning. When φ_s is $\exists x{:}s'.\varphi_1$, we need to prove, by induction hypothesis, that $\Gamma^{\mathsf{MmL}} \vdash \varphi_1^{\mathsf{AML}} \subseteq [\![s]\!] \to \exists x{:}s'.\varphi_1^{\mathsf{AML}} \subseteq [\![s]\!]$, which holds by noting that $x \notin \mathrm{FV}([\![s]\!])$. When φ_s is $\mu X{:}s.\varphi_1$, we need to prove, by induction hypothesis, that $\Gamma^{\mathsf{MmL}} \vdash X \subseteq [\![s]\!] \wedge \varphi_1^{\mathsf{AML}} \subseteq [\![s]\!] \to \mu X.\varphi_1^{\mathsf{AML}} \subseteq [\![s]\!]$. By Deduction Theorem [10, Theorem 92], we need to prove that $\Gamma^{\mathsf{MmL}} \cup \{X \subseteq [\![s]\!]\} \cup \{\varphi_1^{\mathsf{AML}} \subseteq [\![s]\!]\} \vdash \mu X.\varphi_1^{\mathsf{AML}} \subseteq [\![s]\!]$. By simple AML reasoning, we can change $\subseteq$ to $\to$ and the proof obligation becomes $\mu X.\varphi_1^{\mathsf{AML}} \to [\![s]\!]$. By (Knaster-Tarski), we need to prove $\varphi_1^{\mathsf{AML}}[[\![s]\!]/X] \to [\![s]\!]$, which holds by the induction hypothesis on $\varphi_1^{\mathsf{AML}}[[\![s]\!]/X]$, whose *size* (by certain reasonable measurement) is less than that of $\mu X.\varphi_1^{\mathsf{AML}}$. To sum up, we prove by structural induction on φ the conclusion. □

MmL axioms and theories can be handled directly. For an MmL axiom ψ_s, we define its AML translation $\psi_s^{\mathsf{VALID}} \equiv \mathrm{ws}(\psi) \to \psi_s^{\mathsf{AML}} = [\![s]\!]$, which specifies that ψ_s^{AML} matches all elements of sort s if its variables have appropriate sorts. We define $\Omega^{\mathsf{VALID}} = \{\varphi^{\mathsf{VALID}} \mid \varphi \in \Omega\}$ for a set of MmL axioms Ω. Finally, let Γ^{MmL} include all translated axioms, i.e., $\Omega^{\mathsf{VALID}} \subseteq \Gamma^{\mathsf{MmL}}$.

Theorem 27. $\Omega \vdash_{\mathsf{MmL}} \varphi$ *implies* $\Gamma^{\mathsf{MmL}} \vdash \varphi^{\mathsf{VALID}}$.

Proof. We refer readers to [9, Fig. 1] for the proof system of MmL. At a high level, MmL proof system is a many-sorted version of the AML proof system (Fig. 4) where AML's constant symbols are extended to MmL's many-sorted symbols and AML's application contexts are extended to MmL's *symbol contexts* [9, Definition 10].

The proof rules of MmL are also divided into four categories: (many-sorted) FOL reasoning, (many-sorted symbol) frame reasoning, (many-sorted) fixpoint reasoning, and two technical rules. In the following, we carry out mathematical induction on the length of the MmL proof $\Omega \vdash_{\mathsf{MmL}} \varphi$ and show that all MmL proof rules can be derived in AML.

For MmL proof rules that handle many-sorted FOL reasoning, the reasoning is similar to that of MSFOL (Theorem 13). For MmL proof rules that handle many-sorted symbol frame reasoning, we note that every MmL symbol context becomes AML application context after MmL2AML translation. Indeed, for MmL many-sorted symbol $\sigma \in \Sigma_{s_1 \dots s_n, s}$ and MmL symbol context $\sigma(\varphi_1, \dots, \varphi_{i-1}, \Box, \varphi_{i+1}, \dots, \varphi_n)$, its AML translation

$$\sigma\ \varphi_1^{\mathsf{AML}}\ \cdots\ \varphi_{i-1}^{\mathsf{AML}}\ \Box\ \varphi_{i+1}^{\mathsf{AML}}\ \cdots\ \varphi_n^{\mathsf{AML}},$$

is an application context, where $\Box$ appears under i nested (AML's binary) applications. Therefore, MmL frame reasoning proof rules are automatically subsumed in AML. The same reasoning holds for the two technical proof rules of MmL (the last two rules of [9, Fig. 1]). For MmL's (many-sorted)

fixpoint proof rules, we can prove them using the corresponding AML's (unsorted) fixpoint proof rules, noting that the sorts of MmL variables are dropped after the MmL2AML translation. To sum up, we proved by induction that MmL's many-sorted reasoning is preserved after translation to AML. □

G.3 Subsuming MmL Models in AML

As in Section 4.3, we show that MmL structures are subclass of AML structures.

Definition 28. Let (S, V, Σ) be an MmL signature and $(M, \_\bullet\_, \{\sigma_M\}_{\sigma \in \Sigma^{\mathsf{MmL}}})$ be an AML model with $M \vDash \Gamma^{\mathsf{MmL}}$. Its *MmL-restricted structure* is an MmL structure, denoted $M^r = (\{M^r_s\}_{s \in S}, \{\sigma_{M^r}\}_{s \in S})$, consisting of

- a domain $M^r_s = [\![s_M]\!]_M$ for $s \in S$;
- an interpretation $\sigma_{M^r}(a_1, \ldots, a_n) = \sigma_M\, a_1 \cdots a_n$ for $\sigma \in \Sigma_{s_1 \ldots s_n, s}$, $a_1 \in M^r_{s_1}, \ldots, a_n \in M^r_{s_n}$.

Theorem 29. *Let Ω be any MmL theory and M^0 be any MmL model of Ω. There exists an AML model M with $M \vDash \Gamma^{\mathsf{MmL}}$ whose MmL-restricted structure M^r is exactly M^0. In addition, $\Gamma^{\mathsf{MmL}} \vDash \varphi^{\mathsf{VALID}}$ iff $\Omega \vDash_{\mathsf{MmL}} \varphi$.*

Proof. Assume an MmL model $M^0 = (\{M^0_s\}_{s \in S}, \{\sigma_{M^0}\}_{\sigma \in \Sigma})$. We give an explicit construction of the AML model M. We first define its domain, also denoted M, by letting it contain/include all the following elements/sets:

- \$ for definedness and # for domain symbol, as in the proof of Theorem 15;
- S, the sort set;
- M^0_s for all $s \in S$;
- $[M^0_{s_i} \to [M^0_{s_{i+1}} \to [\cdots \to [M^0_{s_n} \to \mathcal{P}(M^0_s)] \ldots]]]$ for all $\Sigma_{s_1 \ldots s_n, s} \neq \emptyset$ and $1 \leq i \leq n$; this is for interpreting symbols and their partial applications; note that powerset semantics of MmL is captured by the co-domain $\mathcal{P}(M^0_s)$.

Next, we define the following interpretations of constants:

- $\lceil \_ \rceil_M = \{\$\}$ and $[\![\_]\!]_M = \{\#\}$;
- $s_M = \{s\}$ for all $s \in S$;
- $\sigma_M = \{\sigma_{M^0}\}$ for all $\sigma \in \Sigma_{s_1 \ldots s_n, s}$; note that $\sigma_{M^0} : M^0_{s_1} \times \cdots \times M^0_{s_n} \to \mathcal{P}(M^0_s)$ is an element in $[M^0_{s_1} \to [M^0_{s_1} \to [\cdots \to [M^0_{s_n} \to \mathcal{P}(M^0_s)] \cdots]]]$ under the curry-uncurry isomorphism.

Next, we define the application in M as follows:

- $\$ \bullet a = M$ for all $a \in M$;
- $\# \bullet s = M_s$ for all $s \in S$; note that $[\![s_M]\!]_M = \# \bullet s = M_s$
- $\sigma \bullet a = \{\sigma(a)\}$ for all $\sigma \in [A \to B]$ if B is a function space; and $\sigma \bullet a = \sigma(a)$ for all B of the form $\mathcal{P}(M^0_s)$, for some $s \in S$.

We split the last case into two cases, depending on if B is a function space. If not, then $\sigma \bullet a = \sigma(a)$ is already a set in $\mathcal{P}(M^0_s)$.

We first verify that $M \vDash \Gamma^{\mathsf{MmL}} \setminus \Omega^{\mathsf{VALID}}$ and prove $M \vDash \Omega^{\mathsf{VALID}}$ later. For the former, (Definedness) and (Nonempty Sort) are satisfied as in Theorem 16. (Arity) for $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ is satisfied, because σ_M is defined as σ_{M^0} and application is interpreted as the normal function application.

Next, we prove that the restricted model M^r is exactly M. By definition, $M^r = (\{M^r_s\}_{s \in S}, \{\sigma_{M^r}\}_{\sigma \in \Sigma})$ is an MmL model defined as follows:

- the domain $M^r_s = [\![s_M]\!]_M$, which equals M^0_s as mentioned, for all $s \in S$;
- the interpretation σ_{M^r}, for $\sigma \in \Sigma_{s_1 \dots s_n, s}$, is defined as $\sigma_{M^r}(a_1, \dots, a_n) = \sigma_M a_1 \cdots a_n$, for all $a_1 \in M^0_{s_1}, \dots, a_n \in M^0_{s_n}$; note that $\sigma_M a_1 \cdots a_n = \{\sigma_{M^0}\} a_1 \cdots a_n = \sigma_{M^0}(a_1, \dots, a_n)$, so we have $\sigma_{M^r}(a_1, \dots, a_n) = \sigma_{M^0}(a_1, \dots, a_n)$, for all $a_1 \in M^0_{s_1}, \dots, a_n \in M^0_{s_n}$.

Therefore, M^r is exactly M^0 up to isomorphism.

Finally, we prove that $\Gamma^{\mathsf{MmL}} \vDash \varphi^{\mathsf{VALID}}$ iff $\Omega \vDash_{\mathsf{MmL}} \varphi$, which immediately implies that $M \vDash \Omega^{\mathsf{VALID}}$. For clarity, we move the proof to Theorem 30.

□

Theorem 30. *Under the notations in the proof of Theorem 29 and the* MmL2AML *translation, any M^r-valuation ρ of MmL derives an M-valuation ρ^{AML} of AML, with $\rho^{\mathsf{AML}}(x^s) = \rho(x{:}s)$ and $\rho^{\mathsf{AML}}(X^s) = \rho(X{:}s)$. Furthermore, $\overline{\rho^{\mathsf{AML}}}(\varphi_s^{\mathsf{AML}}) = \bar{\rho}(\varphi_s)$ for all ρ; and $\Omega^{\mathsf{VALID}} \vDash \varphi_s^{\mathsf{VALID}}$ iff $\Omega \vDash_{\mathsf{MmL}} \varphi_s$ for all Ω.*

Proof. For simplicity, we drop the sort s and write φ_s as φ and φ_s^{AML} as φ^{AML}.

We first prove $\overline{\rho^{\mathsf{AML}}}(\varphi^{\mathsf{AML}}) = \bar{\rho}(\varphi)$ by structural induction on φ.

Suppose $\varphi \equiv x{:}s$. Then $\overline{\rho^{\mathsf{AML}}}((x{:}s)^{\mathsf{AML}}) = \overline{\rho^{\mathsf{AML}}}(x^s) = \{\rho^{\mathsf{AML}}(x^s)\} = \{\rho(x{:}s)\} = \bar{\rho}(x{:}s)$.

Suppose $\varphi \equiv X{:}s$. Then $\overline{\rho^{\mathsf{AML}}}((X{:}s)^{\mathsf{AML}}) = \overline{\rho^{\mathsf{AML}}}(X^s) = \rho^{\mathsf{AML}}(X^s) = \rho(X{:}s) = \bar{\rho}(X{:}s)$.

Suppose $\varphi \equiv \sigma(\varphi_1, \dots, \varphi_n)$. Then $\overline{\rho^{\mathsf{AML}}}((\sigma(\varphi_1, \dots, \varphi_n))^{\mathsf{AML}}) = \overline{\rho^{\mathsf{AML}}}(\sigma \varphi_1^{\mathsf{AML}} \cdots \varphi_n^{\mathsf{AML}})$
$= \sigma_M \overline{\rho^{\mathsf{AML}}}(\varphi_1^{\mathsf{AML}}) \cdots \overline{\rho^{\mathsf{AML}}}(\varphi_n^{\mathsf{AML}}) = \sigma_M \bar{\rho}(\varphi_1) \cdots \bar{\rho}(\varphi_n) = \sigma_{M^r}(\bar{\rho}(\varphi_1), \dots, \bar{\rho}(\varphi_n)) = \bar{\rho}(\sigma(\varphi_1, \dots, \varphi_n))$.

Suppose $\varphi \equiv \varphi_1 \wedge \varphi_2$. Then $\overline{\rho^{\mathsf{AML}}}((\varphi_1 \wedge \varphi_2)^{\mathsf{AML}}) = \overline{\rho^{\mathsf{AML}}}(\varphi_1^{\mathsf{AML}} \wedge \varphi_2^{\mathsf{AML}}) = \overline{\rho^{\mathsf{AML}}}(\varphi_1^{\mathsf{AML}}) \cap \overline{\rho^{\mathsf{AML}}}(\varphi_2^{\mathsf{AML}}) = \bar{\rho}(\varphi_1) \cap \bar{\rho}(\varphi_2) = \bar{\rho}(\varphi_1 \wedge \varphi_2)$.

Suppose $\varphi \equiv \neg\varphi_1$. Then $\overline{\rho^{\mathsf{AML}}}((\neg\varphi_1)^{\mathsf{AML}}) = \overline{\rho^{\mathsf{AML}}}(\neg\varphi_1^{\mathsf{AML}} \wedge [\![s]\!]) = \overline{\rho^{\mathsf{AML}}}((\neg\varphi_1)^{\mathsf{AML}}) \wedge \overline{\rho^{\mathsf{AML}}}([\![s]\!]) = (M \setminus \overline{\rho^{\mathsf{AML}}}(\varphi_1^{\mathsf{AML}})) \cap [\![s_M]\!]_M = [\![s_M]\!]_M \setminus \overline{\rho^{\mathsf{AML}}}(\varphi_1^{\mathsf{AML}}) = M^r_s \setminus \bar{\rho}(\varphi_1) = \bar{\rho}(\neg\varphi_1)$.

Suppose $\varphi \equiv \exists x{:}s.\varphi_1$. Then $\overline{\rho^{\mathsf{AML}}}((\exists x{:}s.\varphi_1)^{\mathsf{AML}}) = \overline{\rho^{\mathsf{AML}}}(\exists x^s{:}s.\varphi_1^{\mathsf{AML}})$
$= \bigcup_{a \in [\![s_M]\!]_M} \overline{\rho^{\mathsf{AML}}[a/x^s]}(\varphi_1^{\mathsf{AML}}) = \bigcup_{a \in M^r_s} \overline{\rho[a/x:s]}(\varphi_1) = \bar{\rho}(\exists x{:}s.\varphi_1)$.

Suppose $\varphi \equiv \mu X{:}s.\varphi_1$. Then $\overline{\rho^{\mathsf{AML}}}((\mu X{:}s.\varphi_1)^{\mathsf{AML}}) = \overline{\rho^{\mathsf{AML}}}(\mu X^s.\varphi_1^{\mathsf{AML}}) = \mu \mathcal{F}^{\rho^{\mathsf{AML}}}_{\varphi_1^{\mathsf{AML}}, X^s}$, where $\mathcal{F}^{\rho^{\mathsf{AML}}}_{\varphi_1^{\mathsf{AML}}, X^s}(A) = \overline{\rho^{\mathsf{AML}}[A/X^s]}(\varphi_1^{\mathsf{AML}})$ for all $A \subseteq M$. Note that for all $A \subseteq M^r_s = [\![s_M]\!]_M \subseteq M$, by inductive hypothesis, $\overline{\rho^{\mathsf{AML}}[A/X^s]}(\varphi_1^{\mathsf{AML}}) = \overline{\rho[A/X:s]}(\varphi_1)$. On the other hand, $\bar{\rho}(\mu X{:}s.\varphi_1) = \mu \mathcal{F}^{\rho}_{\varphi_1, X:s}$ where $\mathcal{F}^{\rho}_{\varphi_1, X:s}(A) = \overline{\rho[A/X:s]}(\varphi_1)$ for all $A \subseteq M^r_s$, so $\mathcal{F}^{\rho}_{\varphi_1, X:s}$ and $\mathcal{F}^{\rho^{\mathsf{AML}}}_{\varphi_1^{\mathsf{AML}}, X^s}$ are equal over M^r_s, and thus have the same least fixpoints.

In conclusion, $\overline{\rho^{\mathsf{AML}}}(\varphi_s^{\mathsf{AML}}) = \bar{\rho}(\varphi_s)$ for all M^r-valuations ρ.

Next, we show that $M^r \vDash_{\mathsf{MmL}} \varphi_s$ iff $M \vDash \varphi_s^{\mathsf{VALID}}$. Recall that $\varphi_s^{\mathsf{VALID}} \equiv \psi \to (\varphi_s^{\mathsf{AML}} = [\![s]\!])$ where $\psi \equiv \bigwedge_{x^{s'} \in \mathrm{FV}(\varphi_s^{\mathsf{AML}})} x^{s'} \in [\![s']\!] \wedge \bigwedge_{X^{s'} \in \mathrm{FV}(\varphi_s^{\mathsf{AML}})} X^{s'} \subseteq [\![s']\!]$.

(Case "if"): Suppose $M \vDash \varphi_s^{\mathsf{VALID}}$ and we want to show that $M^r \vDash_{\mathsf{MmL}} \varphi_s$. Let ρ be any M^r-valuation. Then we know $\rho^{\mathsf{AML}}(\psi) = M$, as ρ^{AML} is derived from ρ and thus it evaluates variables into their corresponding domains. Since $M \vDash \varphi_s^{\mathsf{VALID}}$, we have $\overline{\rho^{\mathsf{AML}}}(\varphi_s^{\mathsf{AML}} = [\![s]\!]) = M$, which implies that $\overline{\rho^{\mathsf{AML}}}(\varphi_s^{\mathsf{AML}}) = \overline{\rho^{\mathsf{AML}}}([\![s]\!])$, i.e., $\bar{\rho}(\varphi_s) = M^r_s$. Since ρ is arbitrary, we have $M^r \vDash_{\mathsf{MmL}} \varphi_s$.

(Case "only if"): Suppose $M^r \vDash_{\mathsf{MmL}} \varphi_s$ and we want to show that $M \vDash \varphi_s^{\mathsf{VALID}}$. Let ρ^* be any M-valuation. There are two cases. If there exists a variable $x^{s'}$ (or $X^{s'}$) such that $\rho^*(x^{s'}) \notin M^r_{s'}$ (or $\rho^*(X^{s'}) \not\subseteq M^r_{s'}$), then $\overline{\rho^*}(\psi) = \emptyset$, and thus $\overline{\rho^*}(\varphi_s^{\mathsf{VALID}}) = M$. If otherwise, then all variables evaluate to their corresponding domain sets, and thus there exists an M^r-valuation ρ such that $\rho^* = \rho^{\mathsf{AML}}$. Then, $\overline{\rho^*}(\varphi_s^{\mathsf{AML}} = [\![s]\!]) = M$ iff $\overline{\rho^*}(\varphi_s^{\mathsf{AML}}) = \overline{\rho^*}([\![s]\!])$ iff $\overline{\rho^{\mathsf{AML}}}(\varphi_s^{\mathsf{AML}}) = M^r_s$ iff $\bar{\rho}(\varphi_s) = M^r_s$, which holds by assumption.

In conclusion, $M \vDash \varphi_s^{\mathsf{VALID}}$ iff $M^r \vDash_{\mathsf{MmL}} \varphi_s$.

Finally, we show that $\Omega^{\mathsf{VALID}} \models \varphi_s^{\mathsf{VALID}}$ iff $\Omega \models_{\mathsf{MmL}} \varphi_s$. Recall that $\Omega^{\mathsf{VALID}} = \Gamma^{\mathsf{MmL}} \cup \{\varphi^{\mathsf{VALID}} \mid \varphi \in \Omega\}$.

(Case "if"): Suppose $\Omega \vdash_{\mathsf{MmL}} \varphi_s$ and we want to show that $\Omega^{\mathsf{VALID}} \vdash \varphi_s^{\mathsf{AML}}$. Consider any AML model M such that $M \vDash \Omega^{\mathsf{VALID}}$. Note that $\Omega^{\mathsf{VALID}} \supseteq \Gamma^{\mathsf{MmL}}$. By Theorem 29, its restricted model M^r is an MmL model. We just proved in the above that $M \vDash \varphi_s^{\mathsf{VALID}}$ iff $M^r \vDash_{\mathsf{MmL}} \varphi_s$ for any φ_s. Since $M \vDash \Omega^{\mathsf{VALID}}$, we have $M^r \vDash_{\mathsf{MmL}} \Omega$, which implies $M^r \vDash_{\mathsf{MmL}} \varphi_s$, which implies $M \vDash \varphi_s^{\mathsf{AML}}$. Since M is arbitrary, we have $\Omega^{\mathsf{VALID}} \vdash \varphi_s^{\mathsf{AML}}$.

(Case "only if"). Suppose $\Omega^{\mathsf{VALID}} \vdash \varphi_s^{\mathsf{VALID}}$ and we want to show that $\Omega \vDash_{\mathsf{MmL}} \varphi_s$. Consider any MmL model M^* such that $M^* \vDash_{\mathsf{MmL}} \Omega$. By Theorem 29, there exists an AML model, say M, whose restricted model is exactly M^*; in other words, $M^* = M^r$. Therefore, we have $M^r \vDash_{\mathsf{MmL}} \Omega$, which is equivalent to $M \vDash \Omega^{\mathsf{VALID}}$ as we just proved. By assumption, $M \vDash \varphi_s^{\mathsf{VALID}}$, which then implies that $M^r \vDash_{\mathsf{MmL}} \varphi_s$. Since M^r is an arbitrary MmL model, we have $\Omega \vDash_{\mathsf{MmL}} \varphi_s$.

□

H Proof of Results about Order-Sorted Algebras (OSA)

Our main theorem for OSA is Theorem 32. We first define the *OSA-restricted models*, similar to Definition 28.

Definition 31. Let $(M, \_\bullet\_, \{\sigma_M\}_{\sigma \in \Sigma^{\mathsf{OSA}}})$ be an AML model of Γ^{OSA}. Its *OSA-restricted model* is an OSA, denoted $A = (\{A_s\}_{s \in S}, \{f_A^{s_1 \dots s_n, s}\}_{f^{s_1 \dots s_n, s} \in F})$, with

- a carrier set $A_s = [\![s_M]\!]_M$ for every $s \in S$;
- an interpretation $f_A^{s_1 \dots s_n, s} \colon A_{s_1} \times \cdots \times A_{s_n} \to A_s$, defined such that $\{f_A^{s_1 \dots s_n, s}(a_1, \dots, a_n)\} = f_M \bullet a_1 \bullet \cdots \bullet a_n$ for all $a_1 \in A_{s_1}, \dots, a_n \in A_{s_n}$.

The above construction of A is well-defined, because its subset relation is enforced by axiom (Subsort) and its subsort-overloaded functions $f_A^{s_1 \dots s_n, s}$ and $f_A^{s_1' \dots s_n', s'}$ coincide on their overlapped part as they are both defined by f_M.

Theorem 32. *$(S, \leq, F)$-OSA are exactly the OSA-restricted Γ^{OSA}-models.*

Proof. All restricted Γ^{OSA}-models w.r.t. $(S, \leq, F)$ are $(S, \leq, F)$-OSA, by definition. We just need to show the other direction. For any $(S, \leq, F)$-OSA, say $A = (\{A_s\}_{s \in S}, \{f_A^{s_1 \dots s_n, s}\}_{f^{s_1 \dots s_n, s} \in F})$, we need find an AML model $M \vDash \Gamma^{\mathsf{OSA}}$, whose restricted model is exactly A. We first define its domain, also denoted M, by letting it contain/include the following elements/sets:

- \$ for definedness and # for domain;
- S, the sort set;
- A_s, for all $s \in S$;
- $[A_{s_i} \to [A_{s_{i+1}} \to [\cdots \to [A_{s_n} \to A_s] \dots]]]$ for all $F_{s_1 \dots s_n, s} \neq \emptyset$ and $1 \leq i \leq n$;

Next, we define following interpretations of constants:

- $\lceil \_ \rceil_M = \{\$\}$ and $[\![\_]\!]_M = \{\#\}$;
- $s_M = \{s\}$ for all $s \in S$;
- $f_M = \{f_A^{s_1' \dots s_n', s'}\}$ for all $f \in F_{s_1 \dots s_n, s}$, where $f^{s_1' \dots s_n', s'}$ denotes the overloaded copy of f with the largest arity (w.r.t. subsorting).

Next, we define the application function in M as the following:

- $\$ \cdot a = M$ for all $a \in M$;
- $\# \cdot s = A_s$ for all $s \in S$; note that $[\![s_M]\!]_M = \# \cdot \{s\} = A_s$;
- $f \cdot a = \{f(a)\}$ for all $f \in [A \to B]$ and $a \in A$.

The unmentioned cases are irrelevant.

Now we verify that M satisfies all axioms in Γ^{OSA}. Indeed, we only need to consider (Subsort), which is satisfied, because $[\![s]\!]_M = A_s \subseteq A_{s'} = [\![s']\!]_M$, for all $s \leq s'$. Therefore, $M \models \Gamma^{\mathsf{OSA}}$.

Finally, we prove that the restricted model M^r is exactly A. By definition, $M^r = (\{M^r\}_{s\in S}, \{f_{M^r}\}_{f\in F})$ is an $(S, \leq, F)$-OSA defined as follows:

- the domain $M^r_s = [\![s_M]\!]_M$, which equals A_s as we showed above, for all $s \in S$;
- the interpretation $f^{s_1\dots s_n,s}_{M^r}$, for every $f^{s_1\dots s_n,s} \in F_{s_1\dots s_n,s}$, is defined such that $\{f^{s_1\dots s_n,s}_{M^r}(a_1,\dots,a_n)\} = f_M a_1 \cdots a_n$ for all $a_1 \in A_1, \dots, a_n \in A_n$; note that $f_M a_1 \cdots a_n = \{f^{s'_1\dots s'_n,s'}_A\} a_1 \cdots a_n = \{f^{s'_1\dots s'_n,s'}_A(a_1,\dots,a_n)\} = \{f^{s_1\dots s_n,s}_A(a_1,\dots,a_n)\}$, because $f^{s_1\dots s_n,s}$ and $f^{s'_1\dots s'_n,s'}$ are subsort overloaded. Thus, we have $f_{M^r}(a_1,\dots,a_n) = f_A(a_1,\dots,a_n)$ for all $a_1 \in A_1, \dots, a_n \in A_n$.

Therefore, M^r is exactly A up to isomorphism. □

I Proof of Results in Section 7

Proposition 33. Under the above axioms, $\vdash [\![List\ Nat]\!] \subseteq [\![List\ Int]\!]$.

Proof. By (Inductive List), we need to prove $\vdash (\mu L.\, nil \vee cons\ [\![Nat]\!]\ L) \subseteq [\![List\ Int]\!]$. By [10], we replace "$\subseteq$" with "$\to$": $\vdash (\mu L.\, nil \vee cons\ [\![Nat]\!]\ L) \to [\![List\ Int]\!]$. By (Knaster Tarski), we need to prove $\vdash nil \vee cons\ [\![Nat]\!]\ [\![List\ Int]\!] \to [\![List\ Int]\!]$. Since $\vdash [\![Nat]\!] \subseteq [\![Int]\!]$, by (Framing), we have $\vdash cons\ [\![Nat]\!]\ [\![List\ Int]\!] \to cons\ [\![Int]\!]\ [\![List\ Int]\!]$. Then by FOL reasoning, we need to prove that $\vdash nil \vee cons\ [\![Int]\!]\ [\![List\ Int]\!] \to [\![List\ Int]\!]$, which is done by (Pre-Fixpoint). □

J Using AML to Reason about Transition Systems

Here we show more details about the dynamic properties/modalities that can be define in AML as patterns; see also [9, 10].

- Cfg is the configuration sort, and $[\![Cfg]\!]$ is matched by all program configurations; In the following, we use φ, ψ to denote configuration patterns, i.e., $\varphi, \psi \subseteq [\![Cfg]\!]$;
- $\neg_{Cfg}\varphi \equiv (\neg\varphi) \wedge [\![Cfg]\!]$ is matched by all *configurations* that do not match φ;
- $\bullet\varphi$, called one-path next, is matched by all states which have a next state matching φ; specifically, $\bullet[\![Cfg]\!]$ is matched by all non-terminating states;
- $\circ\varphi \equiv \neg_{Cfg} \bullet \neg_{Cfg}\varphi$, called all-path next, is matched by all state whose all next states match φ; specifically, $\circ\bot$ is matched by all terminating states;
- $\Diamond\varphi \equiv \mu X.\, \varphi \vee \bullet X$, called (one-path) eventually, is matched by all states who can reach to a state matching φ in finitely many rewrite steps; by unfolding the fixpoint we can prove $\Diamond\varphi = \varphi \vee \bullet\Diamond\varphi = \varphi \vee \bullet\varphi \vee \bullet\bullet\Diamond\varphi = \dots$.
- $\Box\varphi \equiv \neg_{Cfg}\Diamond\neg_{Cfg}\varphi$, called (all-path) always, is matched by states where φ holds now and hereafter on all possible future states;

- $\mathsf{WF} \equiv \mu X.\, \bigcirc X = \mu X.\, \bigcirc \Box X$, called well-founded states, is matched by all states that do not have an infinite execution path;
- $\Diamond_w \varphi \equiv \mathsf{WF} \rightarrow \Diamond \varphi$, called weak eventually, is matched by states that either not well-founded or eventually reaches φ in finitely many steps; it is used to define one-path reachability.

Let us review the invariant claim that we are about to prove:

$$\Phi_{inv} \equiv \forall n{:}Int.\, \forall s{:}Int.\, (\Psi(\texttt{LOOP}, n, s) \wedge n \geq 0) \rightarrow \Diamond_w \Psi(.\,, 0, s + n(n+1)/2)$$

By definition of $\Diamond_w \varphi$, we only need to prove the above invariant for well-founded states, leading us to prove the following (weakened) claim:

$$\Phi'_{inv} \equiv \mu X.\, \bigcirc \Box X \rightarrow \forall n{:}Int.\, \forall s{:}Int.\, (\Psi(\texttt{LOOP}, n, s) \wedge n \geq 0) \rightarrow \Diamond_w \Psi(.\,, 0, s + n(n+1)/2)$$

By (Knaster-Tarski), we need to prove that

$$\Gamma^{\mathsf{IMP}} \vdash \bigcirc\Box\Phi_{inv} \rightarrow \forall n{:}Int.\, \forall s{:}Int.\, (\Psi(\texttt{LOOP}, n, s) \wedge n \geq 0) \rightarrow \Diamond_w \Psi(.\,, 0, s + n(n+1)/2)$$

By FOL reasoning, we can remove the quantifiers $\forall n{:}Int.\, \forall s{:}Int$:

$$\Gamma^{\mathsf{IMP}} \vdash \bigcirc\Box\Phi_{inv} \rightarrow (\Psi(\texttt{LOOP}, n, s) \wedge n \geq 0) \rightarrow \Diamond_w \Psi(.\,, 0, s + n(n+1)/2)$$

By FOL reasoning, we can do a case analysis on $\Psi(\texttt{LOOP}, n, s) \wedge n \geq 0$ and get two cases: (a) $\Psi(\texttt{LOOP}, n, s) \wedge n = 0$, i.e., $\Psi(\texttt{LOOP}, 0, s)$; and (b) $\Psi(\texttt{LOOP}, n, s) \wedge n \geq 1$. For case (a), the loop condition fails and we can execute the program/configuration $\Psi(\texttt{LOOP}, 0, s)$ symbolically till its terminating configuration. For case (b), the loop condition is satisfied, and we execute its loop body symbolically until we reach the loop again, in the new configuration $\Psi(\texttt{LOOP}, n-1, s+n)$. Recall that program execution = AML reasoning, so the above execution process yields a proof of the following AML theorem (recall that it takes 12 steps to consume the body of the loop; see Section 8):

$$\Gamma^{\mathsf{IMP}} \vdash \Psi(\texttt{LOOP}, n, s) \wedge n \geq 1 \rightarrow \bullet^{12} (\Psi(\texttt{LOOP}, n-1, s+n) \wedge n \geq 1)$$

By FOL reasoning, we only need to prove that

$$\Gamma^{\mathsf{IMP}} \vdash \bigcirc\Box\Phi_{inv} \rightarrow \bullet^{12} (\Psi(\texttt{LOOP}, n-1, s+n) \wedge n \geq 1) \rightarrow \Diamond_w \Psi(.\,, 0, s + n(n+1)/2)$$

Now by the "progressive-ness axiom" [10, Proposition 117(18)] and some standard reasoning about the modalities, we have that

$$\begin{aligned} \Gamma^{\mathsf{IMP}} \vdash \bigcirc\Box\Phi_{inv} \wedge \bullet^{12} (\Psi(\texttt{LOOP}, n-1, s+n) \wedge n \geq 1) \\ \rightarrow \Diamond \left(\Phi'_{inv} \wedge \Psi(\texttt{LOOP}, n-1, s+n) \wedge n \geq 1\right) \end{aligned}$$

Then by (Framing), we need to prove that

$$\Gamma^{\mathsf{IMP}} \vdash (\Phi_{inv} \wedge \Psi(\texttt{LOOP}, n-1, s+n) \wedge n \geq 1) \rightarrow \Psi(.\,, 0, s + n(n+1)/2)$$

which can be proved by applying the invariant Φ_{inv} on $\Psi(\texttt{LOOP}, n-1, s+n) \wedge n \geq 1$.