Journal of Logical and Algebraic Methods in Programming 130 (2023) 100810

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

journal homepage: www.elsevier.com/locate/jlamp

Capturing constrained constructor patterns in matching logic

Xiaohong Chen [a,*], Dorel Lucanu [b], Grigore Roşu [a]

[a] Department of Computer Science, University of Illinois Urbana-Champaign, 201 N Goodwin Ave, Urbana, 61801, IL, USA
[b] Alexandru Ioan Cuza University of Iaşi, Bulevardul Carol I, Nr.11, Iaşi, 700506, Romania

ARTICLE INFO

Article history:
Received 31 January 2021
Received in revised form 20 February 2022
Accepted 1 September 2022
Available online 6 September 2022

Keywords:
Constrained constructor patterns
Matching logic
Order-sorted algebras

ABSTRACT

Reachability logic for rewrite theories consists of a specification of system states that are given by constrained constructor patterns, a transition relation that is given by a rewrite theory, and reachability properties expressed as pairs of state specifications. Matching logic has been recently proposed as a unifying foundation for programming languages, specification and verification. It is known that reachability properties can be naturally expressed in matching logic. In this paper, we show that constrained constructor patterns can be faithfully specified as a matching logic theory. As a result, we obtain a full encoding of reachability logic for rewrite theories as matching logic theories, by combining the two encodings. We also show that the main properties of constrained constructor patterns can be specified and proved within matching logic, using the existing proof system.

1. Introduction

Constrained constructor patterns are the bricks of the *rewrite-theory-generic* reachability logic framework [1,2], which generalizes the programming-language-generic reachability logic implemented in the $\mathbb{K}$ framework (https://kframework.org). Rewrite-theory-generic reachability logic framework allows to verify not only conventional programs based on their operational semantics but also concurrent and distributed systems specified by rewrite theories.

Given an order-sorted specification $(\Sigma, E \cup B)$, which is used as support for rewrite theories, the set of equations $E \cup B$ is divided into a set of basic equations B that admits a B-unification algorithm and a set of equations E that can be turned into a set of rewrite rules $\vec{E}$ that are convergent modulo B, assuming that the theory $(\Sigma, E \cup B)$ is sufficiently complete [3]. *Constrained constructor patterns* are defined based on the relationship between two algebras, $T_{\Sigma/E\cup B}$ and $C_{\Omega/E_\Omega,B_\Omega}$, where $T_{\Sigma/E\cup B}$ is an initial $(\Sigma, E \cup B)$-algebra and $C_{\Omega/E_\Omega,B_\Omega}$ is its canonical constructor $(\Omega, E_\Omega \cup B_\Omega)$-algebra, as follows:

1. $T_{\Sigma/E\cup B}$ is isomorphic to the canonical term algebra $C_{\Sigma/E,B}$, consisting of B-equivalence classes of Σ-terms that are irreducible under $\vec{E}$ modulo B;
2. $(\Omega, E_\Omega \cup B_\Omega) \subseteq (\Sigma, E \cup B)$ is a sub-specification for constructors;
3. $C_{\Sigma/E,B}|_\Omega = C_{\Omega/E_\Omega,B_\Omega}$.

Then, a *constrained constructor pattern predicate* has the form $u|\varphi$, where u is a constructor term pattern and φ is a quantifier-free first-order logic (FOL) formula. Constrained constructor patterns are built from constrained constructor pattern predicates and propositional conjunction and disjunction. Semantically, $u|\varphi$ is the set of states $[\![u|\varphi]\!] \subseteq C_{\Omega/E_\Omega,B_\Omega}$ that

* Corresponding author.
E-mail address: xc3@illinois.edu (X. Chen).

https://doi.org/10.1016/j.jlamp.2022.100810
2352-2208/© 2022 Elsevier Inc. All rights reserved.

match term u such that φ holds. That is, for each $a \in [\![u|\varphi]\!]$, there is a valuation ρ such that φ holds, written $\rho \vDash \varphi$, and $a = \rho(u)$.

As an example, consider the constrained process configurations (CPC) that are used, e.g., in the QLOCK mutual exclusion protocol [4,1]. A CPC is represented by a term $\langle n \mid w \mid c \mid q \rangle$ where n, w, c denote the sets of identifiers for normal, waiting, and critical processes, respectively. The constraint constructor pattern $\langle n \mid w \mid c \mid q \rangle \mid dupl(n, w, c) \neq tt$ then states that the union of n, w, and c have no duplicated identifiers.

There are several additional operations over constrained constructor patterns required to express reachability properties and to support their verification in a computational efficient way. These include (parameterized) subsumption, over-approximation of the complements, and parameterized intersections. The definitions of these operations exploit the cases when the matching and unification modulo $E \cup B$ can be efficiently solved, using, e.g., the theory of variants [5,6].

On the other hand, *matching logic* [7–9] is a variant of first-order logic (FOL) with fixpoints that makes no distinction between functions and predicates. It uses instead symbols and application to uniformly build patterns that can represent static structures and logical constraints at the same time. Semantically, matching logic patterns are interpreted as the sets of elements that match them. The functional interpretation is obtained by adding axioms like $\exists y . \mathbb{s}\, x = y$ that forces the pattern $\mathbb{s}\, x$ to be evaluated to a singleton. The conjunction and disjunction are interpreted as the intersection, respectively union. For instance, the matching logic pattern $\exists x{:}Nat . \mathbb{s}\, x \wedge (x = 2 \vee x = 5)$, when interpreted over the natural numbers, denotes the set $\{3, 6\}$ since $\mathbb{s}\, x$ is matched by the successor of x, constants 2 and 5 are matched by the numbers 2 and 5, respectively, and $x = n$ is a "predicate": it matches either the entire carrier set when x and n are matched by the same elements, or otherwise the empty set.

The **main contribution** of this paper is a thorough study on the similarity between constrained constructor patterns and matching logic patterns. Our key insight is the following equivalence relation:

$$\underline{\textit{Constrained Constructor Patterns}} \qquad u \mid \varphi \Longleftrightarrow u \wedge \varphi \qquad \underline{\textit{Matching Logic Patterns}}$$

The constrained constructor pattern on the left-hand side means, intuitively, a system state that matches the term u and satisfies the constraint φ. In matching logic, structures and logical constraints are unified in terms of *patterns*. In other words, both u and φ are patterns, so we can replace the asymmetric notation "|" with the symmetric conjunction "$\wedge$". To *match* $u \wedge \varphi$, one must match both u and φ. Since order-sorted algebras (OSA) can be captured in matching logic [9], we were tempted to think that this equivalence relation is a natural one. However, when we started to formalize this intuition, we realized a few interesting challenges that we need to address:

1. Can the (meta-)theory of constrained constructor patterns developed in [1,2] be expressed as a logical theory (i.e., a set of axioms) in matching logic? This is the main research question addressed in the paper. The answer is given by the results included in Sections 5–7.
2. How to formalize a constructor decomposition of an order-sorted equational theory? The answer is given in Section 6.3.
3. Canonical models are built using convergent rewriting. How the convergent rewriting can be formalized in matching logic? The answer is given in Section 6.2.
4. How to express the equivalence between a constrained constructor pattern and its matching logic encoding? The answer is given by the Theorem 7.4.
5. Which are the most suitable matching logic patterns that capture constrained constructor pattern operations? The answer is given in Section 7.1.1–7.1.4.
6. How the matching logic proof system can be used to reason about constrained constructor patterns? The answer is given by the Propositions 5.9–5.11 and the Theorems 7.5, 7.6, and 7.9.
7. How the existing mechanisms for constrained constructor patterns can be used in matching logic reasoning? The answer is given by the Theorems 7.5, 7.7, and 7.9, which show where the matching and unification algorithms can help in automating the matching logic reasoning.

Our approach is as follows. Since the constrained constructor patterns are heavily-based on canonical model given by the constructors, we define a generic matching logic theory that encode the initial semantics of an order-sorted equational specification. Then, we use this theory as a foundation to encode the canonical model and the other ingredients of the constrained constructor patterns. In order to better understand the relationship between the two approaches, we consider a running example, the constrained process configurations (CPC), and show how to define it in matching logic. This example gives us a better view of the specificity of matching logic axioms and how the OSA canonical model is reflected in matching logic. Since the matching logic axiomatization includes the complete specifications of natural numbers, (finite) lists and (finite) multisets, and it specifies their carrier sets using least fixpoints, we can derive from the specifications an induction proof principle for them.

The establishment of the connection between constrained constructor patterns and matching logic patterns have two benefits. For matching logic, the existing algorithms for constrained constructor patterns bring *more automation* to matching logic reasoning. Since matching, unification, and narrowing modulo theories can all be specified using matching logic patterns, we identify important *fragments* of matching logic that admit good algorithmic and/or semi-algorithmic support. For constrained constructor patterns, our work makes it possible to generate machine-checkable proof objects as correctness certificates for the algorithms and/or semi-algorithms implemented for constrained constructor patterns and rewrite-theory-generic reachability logic framework. Matching logic has a very small proof checker in only 240 lines of code [10]. Since the meta-theory of rewriting logic can be specified *within* matching logic as lemmas and theorems (see, e.g., Section 6), we can formalize the correctness of the matching, unification, and narrowing algorithms as matching logic patterns, whose proofs can be encoded and checked.

Comparison with the conference paper [11] This paper substantially extends the conference paper [11] in the following ways:

- Sections 2–3 were extended in order to make the paper self-contained.
- Section 5 shows how the order-sorted (conditional) equational algebra and its initial semantics is captured in matching logic. This is obtained by extending the approach in [12] for many-sorted logic.
- Sections 5.3–5.5 (the CPC example) were completely rewritten in the terms of the new approach.
- Sections 6 and 7 show a complete encoding of the constrained constructor patterns in matching logic, including the constructor decomposition and the canonical model.
- The proofs of the reported results are included.
- We use an informal specification language, but intuitive, to represent matching logic theories.

Related work Defining constructors using matching logic patterns is firstly discussed in [7]. The later work [12] studies how to define algebras and especially, initial algebras modulo any equations, using matching logic theories. While [12] mainly focuses on many-sorted algebras, in this paper we have extended the approach to defining order-sorted algebras. Reachability logic is shown to be definable in matching logic as a theory in [8]. Matching logic was developed as a logical foundation for the $\mathbb{K}$ framework (https://kframework.org) [13] and a particular kind of constraint constructor patterns are implicitly used in $\mathbb{K}$ for representing the symbolic configurations of the programs [14,15]. Constructor decomposition of order-sorted equational theories together with variant unification algorithms [16] are also used to decide satisfiability for the initial algebras of theory combinations that satisfy certain conditions [6]. That approach can be an inspiration source for finding new decidable fragments of matching logic. A first step for encoding the unification in matching logic has been done in [17].

Structure of the paper We recall the main concepts and results on order-sorted algebras and constrained constructor patterns in Section 2; these are needed to show how they are encoded in matching logic. In Section 3, we introduce matching logic in details, as it was recently proposed. We introduce a couple of matching logic theories needed to capture the order-sorted algebra and its initial semantics in Section 4. In Section 5 we use the CPC example to see the order-sorted matching logic theory at work in practice. In Sections 6 and 7, we describe a matching logic theory that completely specifies the constrained constructor patterns and their operations, including the canonical model. We conclude in Section 8.

2. Constructor decomposition of order-sorted equational theories

We review the basic concepts and key results about *order-sorted algebras*, abbreviated OSAs), following the standard approach in [18–20], and *constructor decomposition of order-sorted equational theories* following [1,2,20].

2.1. Order-sorted algebras

2.1.1. Sorts, signatures, and algebras

We use S to denote a set of *sorts*, denoted s, s_1, s_2, ... A partial ordering $\leq \subseteq S \times S$ is called *a subsort relation*. If $s_1 \leq s_2$, we say that s_1 is a *subsort* of s_2. The subsort relation is extended to $S^* \times S^*$ as follows:

$$s_1 \dots s_n \leq s'_1 \dots s'_n \qquad \text{iff} \qquad s_1 \leq s'_1, \dots, s_n \leq s_n.$$

We define the equivalence relation $\equiv_{\leq}$ by $\equiv_{\leq} = (\leq \cup \leq)^+$. An $\equiv_{\leq}$-equivalence class $[s]$ is called a *connected component*.

We use $F = \{F_{s_1 \dots s_n, s}\}_{s_1,\dots,s_n,s \in S}$ to denote a $(S^* \times S)$-indexed set of *operation symbols* or simply *operations*. For each $f \in F_{s_1 \dots s_n, s}$, we call s_1, ..., s_n the *argument sorts* and s the *return sort*. For $f \in \Sigma_{s_1 \dots s_n, s}$, we also write it using the function notation:

$$f : s_1 \times \dots \times s_n \to s.$$

When $n = 0$, we write $f \in F_{\epsilon, s}$ or $f : \epsilon \to s$ and call it a *constant*. We use $[F]_{s_1 \dots s_n, s}$ to denote the operation set given by

$$[F]_{s_1 \dots s_n, s} = \bigcup_{s'_1 \in [s_1]} \cdots \bigcup_{s'_n \in [s_n]} \bigcup_{s' \in [s]} F_{s'_1 \dots s'_n, s'}.$$

An *order-sorted signature* is a tuple $(S, \leq, F)$. We say that $(S, \leq, F)$ is *sensible* if

$$s_1 \dots s_n \equiv_{\leq} s'_1 \dots s'_n \text{ implies } s \equiv_{\leq} s', \qquad \text{for every } f \in F_{s_1 \dots s_n, s} \cap F_{s'_1 \dots s'_n, s'}.$$

For the simplicity of presentation, we assume that S and F are finite sets. The results and methods in this paper can be applied to infinite sort and operation sets without technical difficulties.

Example 2.1. Let us define an order-sorted signature $(S, \leq, F)$ for lists and multisets of natural numbers (abbreviated NLM) as follows:

$S = \{Nat, List, MSet, NeMSet\}$;

$\leq = \{Nat < List, Nat < NeMSet < MSet\} \cup =_S$;

F includes the following operation symbols:

$\mathbb{0} : \to Nat$, $\mathbb{s}\_ : Nat \to Nat$, $plus : Nat \times Nat \to Nat$;

$nil : \to List$, $\_;\_ : List \times List \to List$, $len : List \to Nat$;

$empty : \to MSet$, $\_\_ : MSet \times MSet \to MSet$,

$\_\_ : NeMSet \times NeMSet \to NeMSet$.

The above signature is sensible. Note that the operation $\_\_$ is overloaded.

Definition 2.2. Given an order-sorted signature $(S, \leq, F)$, an $(S, \leq, F)$-*algebra* or simply an *algebra* A consists of:

- A *carrier set* A_s for each $s \in S$ such that $s \leq s'$ implies $A_s \subseteq A_{s'}$;
- An *operation interpretation* $A_f : A_{s_1} \times \cdots \times A_{s_n} \to A_s$ for each $f : s_1 \times \cdots \times s_n \to s$; such that the following property holds:

For any $f \in F_{s_1 \ldots s_n, s} \cap F_{s'_1 \ldots s'_n, s'}$, $s_1 \equiv_\leq s'_1, \ldots, s_n \equiv_\leq s'_n, s \equiv_\leq s'$

and $(a_1, \ldots, a_n) \in (A_{s_1} \times \cdots \times A_{s_n}) \cap (A_{s'_1} \times \cdots \times A_{s'_n})$

we have $A_{f:s_1 \ldots s_n \to s}(a_1, \ldots, a_n) = A_{f:s'_1 \ldots s'_n \to s'}(a_1, \ldots, a_n)$

Example 2.3. We define an example algebra for NLM in Example 2.1. The carrier sets are given as follows:

- $A_{Nat} = \mathbb{N}$.
- $A_{List} = \mathbb{N}^*$, which is the set of finite sequences over $\mathbb{N}$.
- $A_{MSet} = \mathbb{N}^{\mathbb{N}}$, which is the set of functions from $\mathbb{N}$ to $\mathbb{N}$. We use # to denote such functions, with the intuition that $\#(n)$ is the number of occurrences of n in the multiset.
- $A_{NeMSet} = \mathbb{N}^{\mathbb{N}} \setminus \{\emptyset\}$, where $\emptyset$ is the function that returns 0 on all numbers.
- For every $n \in \mathbb{N}$, we also regard it as the one-element sequence that contains n.
- For every $n \in \mathbb{N}$, we also regard it as the function given as

$$\#_n(m) = \begin{cases} 1 & m = n \\ 0 & m \neq n \end{cases}$$

The last two bullets make sure that $A_{Nat} \subseteq A_{List}$ and $A_{Nat} \subseteq A_{NeMSet} \subseteq A_{MSet}$. The symbol interpretations are given as follows:

- $A_{\mathbb{0}} = 0$.
- $A_{\mathbb{s}}$ is the function given by $A_{\mathbb{s}}(n) = n + 1$ for any $n \in \mathbb{N}$.
- A_{plus} is the function given by $A_{plus}(m, n) = m + n$ for any $m, n \in \mathbb{N}$.
- A_{nil} is the empty sequence.
- $A_{\_;\_}$ is the concatenation of sequences.
- A_{len} returns the length of a sequence.
- $A_{empty} = \emptyset$.
- $A_{\_\_ : MSet \times MSet \to MSet}$ is the union of two multisets.
- $A_{\_\_ : NeMSet \times NeMSet \to NeMSet}$ is the union of two nonempty multisets.

We call the above algebra the *standard algebra* of NLM, since all sorts and operations are interpreted in the expected way.

2.1.2. Terms and term interpretations

Definition 2.4. Let V be a set of *S-sorted variables* $x, y, \ldots$, where the sort of x is denoted $\mathsf{sort}(x) \in S$. Let $T_{F,s}(V)$ be the set of *s-sorted terms with variables in V*, defined inductively by the following grammar (where $x \in V$, $\mathsf{sort}(x) = s$, $f \in F_{s_1 \ldots s_n, s}$, $s' \le s$, and $t_{s_i} \in T_{F,s_i}(V)$ for $1 \le i \le n$):

$$t_s ::= x \mid f(t_{s_1}, \ldots, t_{s_n})$$

Let $T_F(V) = \bigcup_{s \in S} T_{F,s}(V)$ be the set of all terms. The set $T_{F,s} = T_{F,s}(\emptyset)$ includes the *ground terms of sort s* and $T_F = \bigcup_{s \in S} T_{F,s}$ includes all ground terms. An order-sorted signature $(S \le, F)$ is said to have *nonempty sorts* if $T_{F,s} \neq \emptyset$ for each $s \in S$. We say that $(S \le, F)$ is *pre-regular* if each $t \in T_F$ has a *least sort* $\mathit{ls}(t)$, which is the smallest element of $\{s \in S \mid t \in T_{F,s}\}$. In this paper, we assume that all sorts are nonempty and all signatures are pre-regular.

Definition 2.5. A *valuation* $\varrho \colon V \to A$ is a function such that $\varrho(x) \in A_{\mathsf{sort}(x)}$ for all $x \in V$. The corresponding *term interpretation* $\bar{\varrho} \colon T_F(V) \to A$ is defined as expected:

1. $\bar{\varrho}(x) = \varrho(x)$ and
2. $\bar{\varrho}(f(t_{s_1}, \ldots, t_{s_n})) = A_f\left(\bar{\varrho}(t_{s_1}), \ldots, \bar{\varrho}(t_{s_n})\right)$.

Example 2.6. Consider the signature NML in Example 2.1 and its standard algebra A given as Example 2.3. Let ϱ be a valuation such that $\varrho(x) = 0$, $\varrho(y) = 1$, $\varrho(s) = \langle 1, 2, 3\rangle$, $\varrho(X) = \{1, 1, 2\}$, where x and y have sort *Nat*, s has sort *List*, and X has sort *MSet*. The following are the interpretations of some NML terms:

- $\bar{\varrho}(\mathit{plus}(x, y)) = 0 + 1 = 1$.
- $\bar{\varrho}(\mathit{len}(s)) = A_{\mathit{len}}(\langle 1, 2, 3\rangle) = 3$.
- $\bar{\varrho}(\mathit{len}(x)) = A_{\mathit{len}}(\langle 0\rangle) = 1$.
- $\bar{\varrho}(XX) = A_{\_\_}(\bar{\varrho}(X), \bar{\varrho}(X)) = A_{\_\_}(\{1, 1, 2\}, \{1, 1, 2\}) = \{1, 1, 1, 1, 2, 2\}$.
- $\bar{\varrho}(xx) = A_{\_\_}(\bar{\varrho}(x), \bar{\varrho}(x)) = A_{\_\_}(\{0\}, \{0\}) = \{0, 0\}$.

Remark 2.7. If t is a ground term (i.e., $V = \emptyset$), then we write $\mathsf{eval}_A(t)$ or simply $\mathsf{eval}(t)$ to denote the interpretation of t in A.

2.1.3. Equations, equational specifications, equational deduction

Given an order-sorted signature $(S, \le, F)$, an *(unconditional) (F-)equation* is written $\forall V.\, t = t'$, where $t, t' \in T_{F,s}(V)$. A *conditional (F-)equation* is written $\forall V.\, \bigwedge_i t_i = t'_i \to t = t'$. An F-algebra A *validates* an unconditional equation $\forall V.\, t = t'$ (resp., a conditional equation of the form $\forall V.\, \bigwedge_i t_i = t'_i \to t = t'$), if $\bar{\varrho}(t) = \bar{\varrho}(t')$ for all $\varrho \colon V \to A$ (resp. $\bar{\varrho}(t_i) = \bar{\varrho}(t'_i)$ for all $1 \le i \le n$ implies $\bar{\varrho}(t) = \bar{\varrho}(t')$, for all $\varrho \colon V \to A$). For a conditional or unconditional equation e, we write $A \vDash_{\mathsf{Alg}} e$ to indicate that A validates e. We also say that e *holds* in A.

Example 2.8. Let A be the standard algebra for NML, as defined in Example 2.3. The following propositions hold:

- $A \vDash_{\mathsf{Alg}} \forall x, y : \mathit{Nat}.\, \mathit{plus}(x, y) = \mathit{plus}(y, x)$.
- $A \vDash_{\mathsf{Alg}} \forall l_1, l_2 : \mathit{List}.\, \mathit{plus}(\mathit{len}(l_1), \mathit{len}(l_2)) = \mathit{len}(l_1; l_2)$.
- $A \vDash_{\mathsf{Alg}} \forall s_1, s_2 : \mathit{MSet}.\, s_1\, s_2 = s_2\, s_1$.

As we can see, equations can be used to restrict how algebras interpret symbols. Given a set E of equations, called an *equational specification*, we write $A \vDash_{\mathsf{Alg}} E$ to indicate that all (un)conditional equations in E hold in A. In this case, we say that A is an E-algebra.

On the other hands, equations can be used for formal derivation. An *equational deduction system* is a proof system that formally derives equational theorems from an equational specification. There are many equivalent definitions of equational deduction in the literatures. We use the following standard definition.

Definition 2.9. *Equational deduction* is inductively defined by the following 6 equational proof rules:

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

6. substitution: if $E \vdash_{\mathsf{Alg}} \forall V . t = t'$ and $\theta \colon V \to T_F(U)$ then $E \vdash_{\mathsf{Alg}} \forall U . \theta(t) = \theta(t')$.

Remark 2.10. For the case when E includes conditional equations, the rules 1.axiom and 6.substitution must be replaced by

1'&6'. instantiation: if $(\forall V . u_1 = v_1 \wedge \cdots \wedge u_n = v_n \to t = t') \in E$, $\theta \colon V \to T_F(U)$, and $E \vdash_{\mathsf{Alg}} \forall U . \theta(u_1) = \theta(v_1)$, ..., $E \vdash_{\mathsf{Alg}} \forall U . \theta(u_1) = \theta(v_1)$ then $E \vdash_{\mathsf{Alg}} \forall U . \theta(t) = \theta(t')$.

Equational deduction is sound and complete.

Theorem 2.11. *$E \vdash_{\mathsf{Alg}} \forall V . t = t'$ iff $E \vDash_{\mathsf{Alg}} \forall V . t = t'$.*

2.1.4. (Quotient) term algebras and initial algebras

Given an algebra A, a *congruence* R on A is a family of equivalence relations $R_s \subseteq A_s \times A_s$ for each $s \in S$, such that

- $(a_i, b_i) \in R_{s_i}$ for $1 \le i \le n$ implies $\big(A_f(a_1, \dots, a_n), A_f(b_1, \dots, b_n)\big) \in R_s$, for all $f \in F_{s_1 \dots s_n, s}$ and $a_i, b_i \in A_{s_i}$, $1 \le i \le n$; and
- if $s \le s'$ and $a, b \in A_s$, then $(a, b) \in R_s$ iff $(a, b) \in R_{s'}$.

We use $A_{/R}$ to indicate the *R-quotient algebra of A*, defined as follows:

- Carrier set $A_{/R,s} = \{[a]_R \mid a \in A_s\}$ for each $s \in S$, where $[a]_R = \{b \in A_s \mid (a, b) \in R_s\}$ is the R-equivalence class of $a \in A_s$;
- Operator interpretation $A_{/R,f} \colon A_{/R,s_1} \times \cdots \times A_{/R,s_n} \to A_{/R,s}$ for each $f \in F_{s_1 \dots s_n, s}$, defined by $A_{/R,f}([a_1]_R, \dots, [a_n]_R) = [A_f(a_1, \dots, a_n)]_R$ for all $[a_i]_R \in A_{/R,s_i}$, $1 \le i \le n$. The interpretation $A_{/R,f}$ is well-defined because R is a congruence.

Given an order-sorted signature $(S, \le, F)$, its *term algebra* is an algebra whose elements are the ground terms of $(S, \le, F)$ and whose operation interpretations are the operations in F themselves. Formally,

Definition 2.12. The *F-term algebra* is defined as follows:

1. Carrier set $T_s = T_{F,s}$ for $s \in S$, where $T_{F,s}$ is the set of ground terms of sort s.
2. Operator interpretation $T_f \colon T_{F,s_1} \times \cdots \times T_{F,s_n} \to T_{F,s}$ for $f \in F_{s_1 \dots s_n, s}$, defined by $T_f(t_{s_1}, \dots, t_{s_n}) = f(t_{s_1}, \dots, t_{s_n})$ for all $t_{s_i} \in T_{F,s_i}$, $1 \le i \le n$.

We use T_F to also denote the above F-term algebra.

Note that equational deduction generates a congruence on the term algebra T_F.

Proposition 2.13. *Let E be an equational specification. We define $t \simeq_{E,s} t'$ to indicate that $E \vdash_{\mathsf{Alg}} \forall \emptyset . t = t'$, where $t, t' \in T_{F,s}$. Let $\simeq_E$ be the family of relations $\simeq_{E,s}$. Then, $\simeq_E$ is a congruence on the term algebra T_F.*

We abbreviate $[t]_{\simeq_E}$ as $[t]_E$ or simply $[t]$, which denotes the equivalent class of terms that are provably equal to t. We also abbreviate $t \simeq_{E,s} t'$ as $t \simeq_E t'$ or simply $t \simeq t'$. Let $T_{F/E,s} = \big\{[t]_E \mid t \in T_{F,s}\big\}$ and $T_{F/E} = \bigcup_{s \in S} T_{F/E,s}$ be the sets of $\simeq_E$-equivalence classes (of sort s and of any sort).

The (F, E)*-quotient term algebra* is the $\simeq_E$-quotient of the term algebra T_F. Formally,

Definition 2.14. The (F, E)*-quotient term algebra*, written $T_{F/E}$, is defined as follows:

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

Term algebras and quotient term algebras are the canonical, concrete examples of *initial algebras*. We follow the standard definition of initial algebras, which is that of an *initial object* in the category of algebras. We first recall algebra morphisms:

Definition 2.15. For two F-algebras A and B, an *(algebra) morphism* $h \colon A \to B$ is a function such that $h(A_f(a_1, \dots, a_n)) = B_f(h(a_1), \dots, h(a_n))$ for all $f \in F_{s_1 \dots s_n, s}$ and $a_i \in A_{s_i}$, $1 \le i \le n$. In addition, if the inverse $h^{-1} \colon B \to A$ of h exists and is also a morphism, then h is an *isomorphism* and A and B are *isomorphic*, which are regarded as identical.

Definition 2.16. Let (F, E) be an equational specification. An *initial* (F, E)*-algebra* I is one such that for every (F, E)-algebra A, there exists a unique morphism $h_A \colon I \to A$. When $E = \emptyset$, we call I an *initial F-algebra*.

Theorem 2.17. *If F is a sensible order-sorted signature, then the term algebra T_F is the initial F-algebra and the quotient term algebra $T_{F/E}$ is the initial (F, E)-algebra, unique up to isomorphism.*

Equational specification (F, E) states the existence of some data, operations, and equational properties. The initial (F, E)-algebra $T_{F/E}$ is then a "minimal" realization of the specification, in the sense that all its elements are representable by F-terms and it satisfies no other equations except those derived from E.

2.2. Constructor decomposition of order-sorted equational theories

In this section, we review the constructor decomposition of order-sorted equational theories. A constructor decomposition of an order-sorted equational theory is defined based on the following ingredients:

I1. Let $((S, \leq, F), E \cup B)$ be an order-sorted equational theory, where $E \cup B$ is a set of equations partitioned by E and B;

(a) Equations in E are of the form $\psi \to (u = v)$, where u and v have sorts that belong to the same connected component and ψ is a quantifier-free FOL formula built from F-terms and the equality symbol "$=$".
(b) Equations in B are unconditional and regular (i.e., the left- and right-hand sides are linear and include the same variables).

I3. The oriented equation set $\vec{E} = \{\psi \to u \Rightarrow v \mid \psi \to (u = v) \in E\}$, as a rewriting theory, is *convergent modulo* B (i.e., it is sort-decreasing, strictly coherent, and operationally terminating).
I4. The order-sorted signature $(S, \leq, F)$ is B*-pre-regular*, i.e., for each $u = v$ that is an instance of an equation in B, the least sorts of u and v are the same: $ls(u) = ls(v)$.
I5. There is a sub-specification $(\Omega, E_\Omega \cup B_\Omega) \subseteq (F, E \cup B)$ of *constructors*, where $E_\Omega \subseteq E$ and $B_\Omega \subseteq B$.
I6. The specification $((S, \leq, F), E \cup B)$ *protects* $((S, \leq, \Omega), E_\Omega \cup B_\Omega)$, that is,

(a) for all $t, t' \in T_\Omega$, $t =_{B_\Omega} t'$ iff $t =_B t'$,
(b) for all $t \in T_\Omega$, $t = t!_{\vec{E}_\Omega, B_\Omega}$ iff $t = t!_{\vec{E}, B}$, where $t!_{\vec{E}, B}$ is the $\vec{E}$-normal form modulo B of t, and
(c) $C_{F/E,B}|_\Omega = C_{F_\Omega/E_\Omega,B_\Omega}$, where $C_{F/E,B}$ is the canonical term algebra that contains B-equivalences classes of $\vec{E}, B$-irreducible ground F-terms (and similarly $C_{F_\Omega/E_\Omega,B_\Omega}$);

I7. The rewriting theory $((S, \leq, F), B, \vec{E})$ is *sufficiently complete* w.r.t. the signature Ω, i.e., (a) $t!_{\vec{E}, B} \in T_\Omega$ and (b) $u \in T_\Omega$ and $u =_B v$ implies $v \in T_\Omega$.

The states specified by $(F_\Omega, E_\Omega \cup B_\Omega)$ have a minimal representation given by elements in the canonical algebra $C_{F_\Omega/E_\Omega,B_\Omega}$, which are B_Ω-equivalences classes. The symbols in $F \setminus \Omega$ represent operations, over states, specified by $(E \cup B) \setminus (E_\Omega \cup B_\Omega)$. For instance, if a (component of a) state consists of lists, then $(F, E \cup B)$ could specify operations over lists. The partition $E \cup B$ should be given such that the result of the operations can be efficiently computed with $\vec{E}$ rewriting modulo B. The above conditions ensure that the initial $(F, E \cup B)$-algebra $T_{F/E \cup B}$ is isomorphic with the canonical term algebra $C_{\Omega/E_\Omega,B_\Omega}$. Since $E_\Omega \cup B_\Omega$ has fewer equations, it is easier to reason in the canonical term algebra.
A constructor decomposition of an order-sorted equational theory is often written as $(\Omega, E_\Omega \cup B_\Omega) \subseteq (F, E \cup B)$.

3. Matching logic

We review the syntax, semantics, and proof system of matching logic, following [7,8,21,22].

3.1. Matching logic syntax

Matching logic is a uniform logic to specify and reason about mathematical domains, data structures, operations, and even programming languages semantics using formulas called *patterns*. The syntax of patterns is simple and has only 8 constructors (see Definition 3.1), including variables, symbols, application, propositional connectives, $\exists$-quantification, and the μ operator that builds least fixpoints.
In the following, we fix two sets of variables EV and SV. We call variables in EV *element variables*, denoted $x, y, z, \ldots$, and variables in SV *set variables*, denoted $X, Y, Z, \ldots$.

Definition 3.1. A matching logic *signature* Σ is a set whose elements are called *symbols*, denoted $\sigma, \sigma', \sigma_1, \sigma_2, \ldots$. Let Σ be a matching logic signature. Let PATTERN(Σ) or simply PATTERN be the set of Σ-patterns, which are inductively defined by the following grammar:

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

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-hand side of an implication $\varphi_1 \to \varphi_2$. This syntactic requirement is to make sure that φ is monotone with respect to the set X, and thus the least fixpoint $\mu X. \varphi$ exists.

We assume the standard notions of free variables, α-equivalence, and capture-avoiding substitution. Specifically, we use $FV(\varphi)$ to denote the set of (both element and set) variables that are free in φ. We regard α-equivalent patterns as identical. We write $\varphi[\psi/x]$ (resp. $\varphi[\psi/X]$) for the result of substituting ψ for x (resp. X) in φ, where bound variables are implicitly renamed to prevent variable capturing.

The following common constructs can be defined from the basic pattern syntax as syntactic sugar in the usual way:

$$\neg\varphi \equiv \varphi \to \bot \qquad \varphi_1 \vee \varphi_2 \equiv \neg\varphi_1 \to \varphi_2 \qquad \varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$$
$$\top \equiv \neg\bot \qquad \forall x. \varphi \equiv \neg\exists x. \neg\varphi \qquad \nu X. \varphi \equiv \neg\mu X. \neg\varphi[\neg X/X]$$

We assume the standard precedence between the above constructs.

Pattern $\varphi_1\, \varphi_2$ is called application, whose semantics we define in Section 3.2. We assume that application binds tighter than the other constructs, so, for example, $\varphi_1 \wedge \varphi_2\, \varphi_3$ should be understood as $\varphi_1 \wedge (\varphi_2\, \varphi_3)$. We also assume that application is left associative, so we can write $(\cdots((\varphi_1\, \varphi_2)\, \varphi_3) \cdots \varphi_n)$ as $\varphi_1\, \varphi_2\, \varphi_3 \cdots \varphi_n$.

3.2. Matching logic semantics

Matching logic has a *pattern matching semantics*. The semantics of φ in a model M is a set $|\varphi|_M$, whose elements are precisely those that *match* the pattern φ.

In the following, we first define matching logic models.

Definition 3.2. Let Σ be a signature. A Σ-*model*, or simply a model, is a tuple $(M, \_\bullet\_, \{\sigma_M\}_{\sigma\in\Sigma})$ that contains:

- a nonempty carrier set M,
- an application function $\_\bullet\_ \colon M \times M \to \mathcal{P}(M)$ where $\mathcal{P}(M)$ is the power set of M, and
- a symbol interpretation $\sigma_M \subseteq M$ as a subset of M, for each $\sigma \in \Sigma$.

We often use M to denote the above model.

For notational simplicity, we extend $\_\bullet\_$ from over elements to over sets, *pointwisely*, as follows:

$$A \bullet B = \bigcup_{a\in A, b\in B} a \bullet b \ \text{ for } A, B \subseteq M$$

Note that $\emptyset \bullet A = A \bullet \emptyset = \emptyset$ for any $A \subseteq M$.

Definition 3.3. Let M be a model. An M-*valuation*, or simply a valuation, is a mapping $\rho\colon (EV \cup SV) \to (M \cup \mathcal{P}(M))$ such that $\rho(x) \in M$ for $x \in EV$ and $\rho(X) \subseteq M$ for $X \in SV$. In other words, element variables are mapped to elements and set variables are mapped to subsets.

Definition 3.4. Given a model M and a valuation ρ, we define *pattern interpretation* $|\_|_{M,\rho}$ as follows:

- $|x|_{M,\rho} = \{\rho(x)\}$;
- $|X|_{M,\rho} = \rho(X)$;
- $|\sigma|_{M,\rho} = \sigma_M$;
- $|\varphi_1\, \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \bullet |\varphi_2|_{M,\rho}$;
- $|\bot|_{M,\rho} = \emptyset$;
- $|\varphi_1 \to \varphi_2|_{M,\rho} = M \setminus (|\varphi_1|_{M,\rho} \setminus |\varphi_2|_{M,\rho})$;
- $|\exists x. \varphi|_{M,\rho} = \bigcup_{a\in M} |\varphi|_{M,\rho[a/x]}$;
- $|\mu X. \varphi|_{M,\rho} = \mathbf{lfp}\,(A \mapsto |\varphi|_{M,\rho[A/X]})$.

where $\mathbf{lfp}\,(A \mapsto |\varphi|_{M,\rho[A/X]})$ denotes the smallest set A (w.r.t. set inclusion) such that $A = |\varphi|_{M,\rho[A/X]}$, whose existence is guaranteed by the Knaster-Tarski fixpoint theorem [23].

To give the reader better intuition about the semantics of patterns, we discuss two typical types of patterns: predicate patterns and functional patterns.

Predicate patterns Unlike FOL, matching logic patterns can be interpreted as any subsets of the carrier set. How do we represent the truth values "true" and "false" in matching logic? Following the convention in [7], we use the total set M to represent "true" and the empty set $\emptyset$ to represent "false". A pattern whose interpretation is either M or $\emptyset$ is called a *predicate pattern*, or simply a predicate. Predicate patterns correspond to the *formulas* in FOL. They state propositions that either hold or fail.

FOL Reasoning	(TAUTOLOGY)	φ if φ is a propositional tautology over patterns
	(MODUS PONENS)	$\dfrac{\varphi_1 \quad \varphi_1 \to \varphi_2}{\varphi_2}$
	($\exists$-QUANTIFIER)	$\varphi[y/x] \to \exists x.\, \varphi$
	($\exists$-GENERALIZATION)	$\dfrac{\varphi_1 \to \varphi_2}{(\exists x. \varphi_1) \to \varphi_2}$ if $x \notin FV(\varphi_2)$
Frame Reasoning	(PROPAGATION$_\bot$)	$C[\bot] \to \bot$
	(PROPAGATION$_\vee$)	$C[\varphi_1 \vee \varphi_2] \to C[\varphi_1] \vee C[\varphi_2]$
	(PROPAGATION$_\exists$)	$C[\exists x.\, \varphi] \to \exists x.\, C[\varphi]$ if $x \notin FV(C)$
	(FRAMING)	$\dfrac{\varphi_1 \to \varphi_2}{C[\varphi_1] \to C[\varphi_2]}$
Fixpoint Reasoning	(SUBSTITUTION)	$\dfrac{\varphi}{\varphi[\psi/X]}$
	(PREFIXPOINT)	$\varphi[(\mu X.\, \varphi)/X] \to \mu X.\, \varphi$
	(KNASTER-TARSKI)	$\dfrac{\varphi[\psi/X] \to \psi}{\mu X.\, \varphi \to \psi}$
Technical Rules	(EXISTENCE)	$\exists x.\, x$
	(SINGLETON)	$\neg\,(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$

where $C[\varphi]$ indicate a (possibly nested) application pattern of φ.

Fig. 1. Matching logic proof system.

Functional patterns Functional patterns correspond to the *terms* in FOL. A pattern is a functional pattern if its interpretation is a singleton set, including exactly one element. Since a functional pattern denotes, or is matched by, exactly one element, we often blur the distinction between elements (such as a) and singletons (such as $\{a\}$). The simplest functional pattern is an element variable x. More interesting examples can be built from symbols and application.

Fixpoint patterns Given a model M and a valuation ρ, the fixpoint pattern $\mu X.\, \varphi$ yields a function $\mathcal{F} \colon \mathcal{P}(M) \to \mathcal{P}(M)$, defined by $\mathcal{F}(A) = |\varphi|_{\rho[A/X]}$ for all $A \subseteq M$. By the requirement that φ is positive in X (Definition 3.1), we can prove that $\mathcal{F}$ is a monotone function, whose unique least fixpoint $\mu\mathcal{F} \subseteq M$ is exactly the valuation $|\mu X.\, \varphi|_{M,\rho}$. Therefore, $\mu X.\, \varphi$ is a direct, logical incarnation of the least fixpoints in powersets into matching logic patterns. More interestingly, the semantic rule in Definition 3.4 inspires an *induction principle* for reasoning about fixpoint patterns, which we discuss informally here and formalize in Section 3.3 as the proof rule (KNASTER-TARSKI). By Definition 3.4, to show that $|\mu X.\, \varphi|_{M,\rho} \subseteq A$ for a set A, one only needs to show that $|\varphi|_{M,\rho[A/X]} \subseteq A$; or written syntactically, to prove that $(\mu X.\, \varphi) \to \psi_A$ for a pattern/property ψ_A, one only needs to prove that $\varphi[\psi_A/X] \to \psi_A$. This general form of inductive reasoning is included in Section 3.3.

3.3. Matching logic proof system

Definition 3.5. A *matching logic theory* (Σ, Γ) consists of a signature Σ and a set Γ of Σ-patterns, called *axioms*. We say that a model M *validates* (Σ, Γ), written $M \vDash \Gamma$, if for all $\psi \in \Gamma$ and valuations ρ, $|\psi|_{M,\rho} = M$. In other words, M validates all axioms in Γ.

In the literatures, matching logic theories are also called matching logic specifications. A matching logic theory has two aspects. On one hand, it restricts the models by enforcing them to validate the axioms, as in Definition 3.5. On the other hand, it can be used to derive formal theorems that hold for all models of the specification by using the matching logic proof system.

Matching logic has a *fixed* Hilbert-style proof system that supports formal reasoning for all specifications Γ, shown in Fig. 1. We write $\Gamma \vdash \varphi$ to mean that φ is a provable pattern from the axioms in Γ.

In the following, we review some important meta-theorems that can be proved about the proof system so as to give intuition about the types of formal reasoning that are supported in matching logic.

Proposition 3.6. *Let Γ be any specification. Then, the following propositions hold:*

1. *$\Gamma \vdash \varphi$, if φ is a propositional tautology over patterns;*
2. *$\Gamma \vdash \varphi_1$ and $\Gamma \vdash \varphi_1 \to \varphi_2$ imply $\Gamma \vdash \varphi_2$;*
3. *$\Gamma \vdash \varphi[y/x] \to \exists x.\, \varphi$;*

```
theory DEF
Symbols:   def
Notations: ⌈φ⌉ ≡ def φ
Axioms:    (DEFINEDNESS)  ∀x. ⌈x⌉
Notations:
  ⌊φ⌋ ≡ ¬⌈¬φ⌉              // totality
  φ1 = φ2 ≡ ⌊φ1 ↔ φ2⌋      // (true) equality
  φ1 ⊆ φ2 ≡ ⌊φ1 → φ2⌋      // set inclusion
  x ∈ φ ≡ x ⊆ φ            // membership
endtheory
```

Fig. 2. Matching logic theory DEF.

4. $\Gamma \vdash \varphi_1 \to \varphi_2$ *and* $y \notin FV(\varphi_2)$ *imply* $\Gamma \vdash (\exists y.\, \varphi_1) \to \varphi_2$*;*
5. $\Gamma \vdash \varphi = \varphi$*;*
6. $\Gamma \vdash \varphi_1 = \varphi_2$ *and* $\Gamma \vdash \varphi_2 = \varphi_3$ *imply* $\Gamma \vdash \varphi_1 = \varphi_3$*;*
7. $\Gamma \vdash \varphi_1 = \varphi_2$ *implies* $\Gamma \vdash \varphi_2 = \varphi_1$*;*
8. $\Gamma \vdash \varphi_1 = \varphi_2$ *implies* $\Gamma \vdash \psi[\varphi_1/x] = \psi[\varphi_2/x]$*, known as the Leibniz's law of equality.*
9. $\Gamma \vdash (\mu X.\, \varphi) = \varphi[\mu X.\, \varphi/X]$*;*
10. $\Gamma \vdash \varphi[\psi/X] \to \psi$ *implies* $\Gamma \vdash (\mu X.\, \varphi) \to \psi$*; this proof rule is denoted* (KNASTER-TARSKI)*;*

where for (5)-(9) we naturally require that Γ *defines equality (see Section 4.1).*

Properties (1)-(4) capture standard FOL reasoning, (5)-(8) capture standard equational reasoning, and (9)-(10) provide standard fixpoint reasoning (cf. [24, Section 5]). Particularly, rule (9) states that $\mu X.\, \varphi$ is a fixpoint, and thus it is the same after unfolding. Rule (10) characterizes an induction principle about $\mu X.\, \varphi$, following the discussion in Section 3.2. As expected, $\vdash$ is sound:

Theorem 3.7. *For any* Γ *and* φ*, we have* $\Gamma \vdash \varphi$ *implies* $\Gamma \vDash \varphi$*.*

4. Basic matching logic theories

We define some basic matching logic theories in this section.

4.1. Equality and membership

We define a matching logic theory for equality. Specifically, we define a predicate pattern $\varphi = \varphi'$ that holds (i.e., equals to $\top$) iff φ equals to φ', in the sense that they are matched by the same elements. To do that, we introduce an important mathematical instrument called *definedness*. A definedness pattern $\lceil \varphi \rceil$ is a predicate pattern that holds iff φ is *defined*, that is, it is matched by at least one element.

In Fig. 2, we define a matching logic theory DEF that defines one symbol (written *def*) and one axiom called (DEFINEDNESS). The definedness pattern $\lceil \varphi \rceil$ is defined as syntactic sugar of the application pattern ($def\ \varphi$), where $def \in \Sigma$ is a matching logic symbol. Intuitively, (DEFINEDNESS) states that every element x is defined. Indeed, x is always matched by one element and thus is *defined*, by definition.

Totality, on the other hand, is defined as $\lfloor \varphi \rfloor \equiv \neg \lceil \neg \varphi \rceil$. It holds whenever φ is matched by all elements, that is, it equals to $\top$. Equality $\varphi_1 = \varphi_2$ and set inclusion $\varphi_1 \subseteq \varphi_2$ are defined from totality. Membership $x \in \varphi$ has the same meaning as $x \subseteq \varphi$, but we still write $x \in \varphi$ as it fits well the intuition that x is an element in φ.

4.2. Natural numbers

We define a theory for natural numbers. The highlight of this definition is that we use the least fixpoint operator μ to specify the set of natural numbers as the smallest set closed under $\mathbb{0}$ and *succ*, which are the two constructors of natural numbers.

In Fig. 3, we define the matching logic theory NN for natural numbers. NN imports DEF and defines three new symbols: $\mathbb{N}$ denotes the set of natural numbers; *zero* denotes the number zero; and *succ* denotes the successor function. Axiom (NAT ZERO) states that *zero* is matched by exactly one element in $\mathbb{N}$. Axiom (NAT SUCC) states that for any x in $\mathbb{N}$, (*succ x*) is an element in $\mathbb{N}$. Axioms (NAT SUCC.1) and (NAT SUCC.2) state that *zero*, (*succ zero*), (*succ succ zero*), ... are all distinct. Finally, (NAT DOMAIN) defines $\mathbb{N}$ as the smallest set D that includes *zero* and is closed under *succ*, using the least fixpoint operator μ. Note that we have a 1-1 correspondence between the NN axioms and the Peano axioms (see, e.g., https://www.britannica.com/science/Peano-axioms):

theory NN Imports: DEF
Symbols: $\mathbb{N}, zero, succ$
Axions:

(Nat Zero)	$\exists x. x \in \mathbb{N} \wedge zero = x$
(Nat Succ)	$\forall x. x \in \mathbb{N} \rightarrow \exists y. y \in \mathbb{N} \wedge succ\ x = y$
(Nat Succ.1)	$succ\ zero \neq zero$
(Nat Succ.2)	$\forall x. \forall y. x \in \mathbb{N} \wedge y \in \mathbb{N} \rightarrow succ\ x = succ\ y \rightarrow x = y$
(Nat Domain)	$\mathbb{N} = \mu D. zero \vee succ\ D$

endtheory

Fig. 3. Matching logic theory NN.

theory SORT Imports: DEF
Symbols: *inh*, *Sort*
Notations:

$\top_s \equiv inh\ s$	// inhabitants of sort s
$\neg_s \varphi \equiv (\neg\varphi) \wedge \top_s$	// negation within sort s
$\forall x : s. \varphi \equiv \forall x. x \in \top_s \rightarrow \varphi$	// $\forall$ within sort s
$\exists x : s. \varphi \equiv \exists x. x \in \top_s \wedge \varphi$	// $\exists$ within sort s
$\varphi : s \equiv \exists z : s. \varphi = z$	// "typing"
$\forall x_1, \ldots, x_n : s. \varphi \equiv \forall x_1 : s. \ldots \forall x_n : s. \varphi$	// multiple $\forall$ within sort s
$\exists x_1, \ldots, x_n : s. \varphi \equiv \exists x_1 : s. \ldots \exists x_n : s. \varphi$	// multiple $\exists$ within sort s

endtheory

Fig. 4. SORT.

1. 0 is a natural number.
2. Every natural number has a successor in the natural numbers.
3. 0 is not the successor of any natural number.
4. If the successors of two natural numbers are the same, then the two original numbers are the same.
5. If the successors of two natural numbers are the same, then the two original numbers are the same.

As we can see, symbols *zero* and *succ* behave like normal FOL functions that build *terms*. This intuition is important so we formalize it in detail. Consider a model $M \vDash$ NN, where $\_\bullet\_ : M \times M \rightarrow \mathcal{P}(M)$ is the application function, $\_\bar{\bullet}\_$ is its pointwise extension, and $\mathbb{N}_M, zero_M, succ_M \subseteq M$ are symbol interpretations. Then, M defines all natural numbers built from $\mathbb{0}$ and *succ in the following sense*:

Proposition 4.1. *The following hold for $M \vDash$ NN:*
1. *By axiom* (Nat Zero)*: $zero_M$ is a singleton, whose (unique) element we denote as $M_{zero} \in \mathbb{N}_M$;*
2. *By* (Nat Succ)*: for any $m \in \mathbb{N}_M$ there exists $next_m \in \mathbb{N}_M$ with $succ_M \bar{\bullet} \{m\} = \{next_m\}$; let $M_{succ} : \mathbb{N}_M \rightarrow \mathbb{N}_M$ be the (unique) function such that $M_{succ}(m) = next_m$;*
3. *By* (Nat Succ.1) *and* (Nat Succ.2)*: $M_{succ}(M_{zero}) \neq M_{zero}$ and M_{succ} is injective;*
4. *By* (Nat Domain)*: $\mathbb{N}_M = \{M_{zero}, M_{succ}(M_{zero}), \ldots\}$;*
5. *Thus, $(\mathbb{N}_M, M_{zero}, M_{succ})$ is standard natural numbers.*

As a notational convention, we write $zero_M$ and $succ_M$ for the (set-theoretic) symbol interpretations and M_{zero} and M_{succ} for the FOL-style functions derived as above. We shall revisit this convention when we define sorts in Section 4.3.

4.3. Sorts

Matching logic is unsorted, but there is a systematic and extensible way to axiomatically define arbitrary sorting structures, following the "sort-as-predicate" paradigm.

In matching logic, a sort s is associated with its *inhabitants*, which are elements "of sort s". Formally, let s be a symbol that represents the *name* of the sort. We define a symbol *inh*, called the *inhabitant symbol*, and use the application pattern (*inh* s), written $\top_s$, to obtain the actual inhabitant set of s.

In Fig. 4, we define the matching logic theory SORT that provides support for handling sorts. SORT does not define any concrete sorts but only the generic infrastructure. It defines *inh* and also a symbol *Sort*, which is the set of all sort names. Then, it introduces some common notations for sorts, such as sorted negation and sorted quantification. In particular, the "typing" construct $\varphi : s$ states that φ is an element of sort s.

SORT is a generic theory that is imported whenever simple sorts are desired. Note that the actual meaning of an element having a sort is open for interpretation and is entirely determined by (user-provided) axioms.

```
theory NAT Imports: NN, SORT
Symbols: Nat, plus, mult, NzNat
Axioms:
```

(NAT SORT)	$Nat : Sort$
(NAT)	$\top_{Nat} = \mathbb{N}$
(NAT PLUS.1)	$\forall x : Nat.\, plus\ x\ zero = x$
(NAT PLUS.2)	$\forall x, y : Nat.\, plus\ x\ (succ\ y) = succ\ (plus\ x\ y)$
(NAT MULT.1)	$\forall x : Nat.\, mult\ x\ zero = zero$
(NAT MULT.2)	$\forall x, y : Nat.\, mult\ x\ (succ\ y) = plus\ x\ (mult\ x\ y)$
(NZNAT SORT)	$NzNat : Sort$
(NZNAT)	$\top_{NzNat} = succ\ \top_{Nat}$

Notations:

$0 \equiv zero,\quad 1 \equiv succ\ 0,\quad 2 \equiv succ\ 1,\quad \cdots$

endtheory

Fig. 5. NAT.

In this and the rest subsections, we define *compound sorts*, such as those for pairs, tuples, and functions. Their specifications are self-explanatory, so we do not get into too much technical detail. At the end of the section, we put all specifications together and define a "prelude" SORT^{++} theory, which is imported when we define order-sorted specifications in Section 5.

4.3.1. The sort of natural numbers

We extend the theory NN for natural numbers by adding an explicit sort *Nat*, as in Fig. 5. Intuitively, (NAT SORT) states that *Nat* is an inhabitant of *Sort*. (NAT) states that the inhabitant set of *Nat* equals $\mathbb{N}$, which is the set of natural numbers defined in Section 4.2. The other four axioms define the Peano addition and multiplication of natural numbers, where we use the sorted quantification (Specification 4).

Recall that in NN, we already define $\mathbb{N}$ as the set of all natural numbers and the two constructors *zero* and *succ*. Therefore, we simply let NAT import NN. Axiom (NAT SORT) specifies that *Nat* is a sort. Axiom (NAT) specifies its inhabitant using $\mathbb{N}$. The remaining four axioms are the standard axioms that define the addition and multiplication of natural numbers, We point out that some axioms in the original specification NN can be rewritten using the notations defined in SORT to simpler forms as follows:

(NAT ZERO)	$zero : Nat$
(NAT SUCC)	$\forall x : Nat.\, (succ\ x) : Nat$
(NAT SUCC.2)	$\forall x, y : Nat.\, succ\ x = succ\ y \to x = y$

Remark 4.2. We point out that the sorted quantification axioms do not *restrict* $\mathbb{s}$ to be only applicable within *Nat*. The pattern $\mathbb{s}\,x$ when x is outside the domain of *Nat* is still a well-formed pattern, whose semantics is not specified by the theory of natural numbers, but can be specified by other theories. For example, the theory of real numbers may re-use $\mathbb{s}$ and overload it as the increment-by-one function on reals. The theory of bounded arithmetic may re-use and overload $\mathbb{s}$ as the successor "function", which is actually a partial function and is undefined on the maximum value. The theory of transition systems may re-use and overload $\mathbb{s}$ as the successor "function", which is actually the underlying transition relation, and $\mathbb{s}\,x$ yields the set of all next states of the state x. In the last two cases, $\mathbb{s}$ is no longer a function because it is not true that $\mathbb{s}\,x$ always returns one element. Therefore, if we use not the sorted quantification axiom but the unsorted one, we cannot re-use $\mathbb{s}$ in the theories of bounded arithmetic or transition systems, without introducing inconsistency. Thus, by using sorted quantification for $\mathbb{s}$ in the theory of natural numbers, we do not restrict but actually encourage the re-use and overloading of $\mathbb{s}$ in other theories. On the other hand, ML is expressive enough if one wants to allow a restricted use of a symbol. For instance, if we want to restrict the use of $\mathbb{s}$ only to *Nat*, then we can add the axiom $\forall x.\, \lceil \mathbb{s}\,x \rceil \to x \in \top_{Nat}$.

Theory NAT captures precisely natural numbers in the following sense. Let M be any model that validates NAT, i.e., $M \vDash \text{NAT}$. Let $M_{Nat} = |\top_{Nat}|_M$ be the inhabitant set of *Nat* in M. Let $\mathbb{0}_M$ be the interpretation of $\mathbb{0}$ in M. By axiom (NAT ZERO), $\mathbb{0}_M$ is a singleton set, whose only element we (ambiguously) denote as $\mathbb{0}$; we have $\mathbb{0} \in M_{Nat}$. For any $n \in M_{Nat}$, variable x, and valuation ρ with $\rho(x) = n$, $|succ\ x|_{M,\rho}$ is a singleton set by axiom (NAT SUCC), whose only element we denote as $\mathbb{s}(n)$; in other words, $\mathbb{s} \colon M_{Nat} \to M_{Nat}$ is a function. Similarly, *plus* and *mult* yield two functions $\mathbb{plus}, \mathbb{mult} \colon M_{Nat} \times M_{Nat} \to M_{Nat}$, respectively.

Proposition 4.3. *Under the above notations,* $(\mathbb{N}, 0, 1, +, \times) \cong (M_{Nat}, \mathbb{0}, \mathbb{s}(\mathbb{0}), \mathbb{plus}, \mathbb{mult})$.

NAT is an important example that shows how to use the generic sorting infrastructure in SORT to axiomatically define sorts and operations. The sort *Nat* of natural numbers will also be imported and used by the specification of tuples (Section 4.3.3) to define tuple projection. It is tempting to "generalize" the notation $\varphi : s$ in SORT to something like $\varphi : s_1 \times \cdots \times s_n \to s$, so that axioms like (Nat Succ) above, which are very common, can be written more compactly and

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

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

endtheory

Fig. 6. PAIR.

intuitively as $succ : Nat \to Nat$. However, we refrain from doing it now because there is a better solution, based on *function sorts*, which we discuss in Section 4.3.4.

Finally, axiom (Nat Domain) is an example of capturing inductive domains using least fixpoint patterns. Our definition of the initial algebra semantics, which we start to define in Section 5, shares the same idea with (Nat Domain).

NAT is self-explanatory. With the notations introduced in SORT, the axioms are highly readable and there is little "encoding cost" to handle sorts in (unsorted) matching logic.

Like in Section 4.2, we define $M_{Sort} = |\top_{Sort}|_M$ and $M_{Nat} = |\top_{Nat}|_M$ as the carriers of *Sort* and *Nat*, respectively. This way, $M_{zero} \in M_{Nat}$ and $M_{succ} \colon M_{Nat} \to M_{Nat}$ build all natural numbers; M_{plus} and M_{mult} (defined similarly) are the standard natural number addition and multiplication.

Besides *Nat*, we also define a sort *NzNat* for nonzero (i.e., positive) natural numbers. Its axiom, (NzNat), has a strong matching logic flavor, where we apply *succ* to the entire carriers of *Nat*. Finally, we define the natural numbers $0, 1, 2, \ldots$ as notations so we can use them as is.

4.3.2. Sorts of pairs

For any sorts s_1 and s_2, we define the *pair sort*, written $Pair\ s_1\ s_2$ or $s_1 \otimes s_2$. Here, *Pair* is a symbol that serves as a *sort constructor*. We also define *pair* as the pair constructor and *fst*, *snd* as pair destructors. The pair of x_1 and x_2 is computed by $pair\, x_1\, x_2$, often written $\langle x_1, x_2 \rangle$. These are formalized in the matching logic theory given in Fig. 6, denoted PAIR.

Intuitively, $\langle x_1, x_2 \rangle$ is the pair of x_1 and x_2. If x_1 has sort s_1 and x_2 has sort s_2, then by (Pair), $\langle x_1, x_2 \rangle$ is an element of sort $s_1 \otimes s_2$. (Pair Sort) states that $s_1 \otimes s_2$ is a sort when s_1 and s_2 are sorts in *Sort*. Hence, we can have nested product sorts such as $s_1 \otimes (s_2 \otimes s_3)$ where s_1, s_2, s_3 are sorts in *Sort*. This allows us to define (finite) *tuple sorts* (Section 4.3.3). (Pair Fst) and (Pair Snd) define the destructors *fst* and *snd*. (Pair Inj) states that two pairs are equal only if their corresponding components are equal. Finally, (Pair Domain) states that the inhabitant set of $s_1 \otimes s_2$ is the set of all pairs $\langle x_1, x_2 \rangle$, with x_1 of sort s_1 and x_2 of sort s_2.

Like in Propositions 4.1 and 4.3, we formalize the above intuition by considering a model $M \vDash$ PAIR, where $pair_M, fst_M, snd_M, Pair_M \subseteq M$ are the corresponding symbol interpretations. Recall that we let $M_{Sort} = M_{inh} \mathbin{\bar{\cdot}} Sort_M$ to be the inhabitant set of *Sort* that includes all sort names in M. Then for each $s \in M_{Sort}$, let the set $M_s = M_{inh} \mathbin{\bar{\cdot}} \{s\}$ be the inhabitant set of s in M.

Proposition 4.4. *Under the above conditions and notations, the following properties hold:*

1. *By* (Pair Sort)*: for $s_1, s_2 \in M_{Sort}$, $Pair_M \mathbin{\bar{\cdot}} \{s_1\} \mathbin{\bar{\cdot}} \{s_2\}$ is a singleton, whose (unique) element we denote $s_1 \otimes_M s_2$; then we have $s_1 \otimes_M s_2 \in M_{Sort}$; intuitively, $s_1 \otimes_M s_2$ is the pair sort of s_1 and s_2;*
2. *By* (Pair)*: for $s_1, s_2 \in M_{Sort}$, $x_1 \in M_{s_1}$, and $x_2 \in M_{s_2}$, $pair_M \mathbin{\bar{\cdot}} \{x_1\} \mathbin{\bar{\cdot}} \{x_2\}$ is a singleton, whose (unique) element we denote $\langle x_1, x_2 \rangle_M$; then we have $\langle x_1, x_2 \rangle_M \in M_{s_1 \otimes_M s_2}$;*
3. *Like in Propositions 4.2 and 4.3.1, let $M_{fst} \colon M_{s_1 \otimes_M s_2} \to M_{s_1}$ and $M_{snd} \colon M_{s_1 \otimes_M s_2} \to M_{s_2}$ be such that $fst_M \mathbin{\bar{\cdot}} \{y\} = \{M_{fst}(y)\}$ and $snd_M \mathbin{\bar{\cdot}} \{y\} = \{M_{snd}(y)\}$, for any $s_1, s_2 \in M_{Sort}$ and $y \in M_{s_1 \otimes_M s_2}$;*
4. *By* (Pair Fst/Snd/Inj)*: for $s_1, s_2 \in M_{Sort}$, $x_1, y_1 \in M_{s_1}$, and $x_2, y_2 \in M_{s_2}$, we have $M_{fst}(\langle x_1, x_2 \rangle_M) = x_1$, $M_{snd}(\langle x_1, x_2 \rangle_M) = x_2$, and $\langle x_1, x_2 \rangle_M = \langle y_1, y_2 \rangle_M$ implies $x_i = y_i$, $(i = 1, 2)$;*
5. *$M_{s_1 \otimes_M s_2} = \{\langle x_1, x_2 \rangle_M \mid x_1 \in M_{s_1}, x_2 \in M_{s_2}\}$;*
6. *Then, $M_{s_1 \otimes_M s_2}$ is exactly the Cartesian product $M_{s_1} \times M_{s_2}$ of M_{s_1} and M_{s_2}, for $s_1, s_2 \in M_{Sort}$.*

4.3.3. Sorts of tuples

Tuples are defined as nested pairs. Let TUPLE be the matching logic theory given in Fig. 7, which imports PAIR and defines a symbol *proj* called *projection*, taking as input a number n and a tuple and returning its nth component. Formally, $\pi_i\ \varphi \equiv proj\ i\ \varphi$, for $i \geq 1$, where we use the notations for natural numbers defined in the theory NAT given in Fig. 5.

theory TUPLE Imports: PAIR, NAT
Symbols: *proj*
Notations:
$\langle \varphi_1, \varphi_2, \ldots, \varphi_n \rangle \equiv \langle \varphi_1, \langle \varphi_2, \ldots, \varphi_n \rangle \rangle$
$s_1 \otimes s_2 \otimes \cdots \otimes s_n \equiv s_1 \otimes (s_2 \otimes \cdots \otimes s_n)$
Axioms: // all axioms are quantified by "$\forall s_1, s_2 : Sort$"
(Proj First) $\forall x_1 : s_1 . \forall x_2 : s_2 . (proj\ 1\ \ \langle x_1, x_2 \rangle) = x_1$
(Proj Rest) $\forall x_1 : s_1 . \forall x_2 : s_2 . \forall n : NzNat.$
$(proj\ (succ\ n)\ \ \langle x_1, x_2 \rangle) = proj\ n\ x_2$
endtheory

Fig. 7. TUPLE.

theory FUNCTION Imports: SORT
Symbols: *Function*
Notations: $s_1 \ominus s_2 \equiv Function\ s_1\ s_2$
Axioms: // all axioms are quantified by "$\forall s_1, s_2 : Sort$"
(Func Sort) $(s_1 \ominus s_2) : Sort$
(Func Domain) $\top_{s_1 \ominus s_2} = \exists f . f \wedge (\forall x : s_1 . (f\ x) : s_2)$
(Func Ext) $\forall f, g : s_1 \ominus s_2 . (\forall x : s_1 . f\ x = g\ x) \to f = g$
endtheory

Fig. 8. FUNCTION.

Table 1
Handling functions in matching logic.

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

Remark 4.5. Following the notations in Proposition 4.4, for a model $M \vDash$ TUPLE, we write $\langle x_1, \ldots, x_n \rangle_M \in M_{s_1 \otimes_M \cdots \otimes_M s_n}$ for the tuple of $x_1 \in M_{s_1}, \ldots, x_n \in M_{s_n}$. Then, $M_{s_1 \otimes_M \cdots \otimes_M s_n}$ is exactly the Cartesian product $M_{s_1} \times \cdots \times M_{s_n}$ of sets $M_{s_1}, \ldots, M_{s_n}$, for $s_1, \ldots, s_n \in M_{Sort}$.

4.3.4. Sorts of functions

For any sorts s_1 and s_2, we define the *function sort*, written $Function\ s_1\ s_2$ or $s_1 \ominus s_2$, where *Function* is a symbol that serves as a sort constructor. Let FUNCTION be Specification 8.

Note that we do not explicitly define the constructors of function sorts. Instead, we axiomatize the *behaviors* of a function. Indeed, axiom (Func Domain) states that a "function" f of sort $s_1 \ominus s_2$ is one such that for any x of sort s_1, $(f\ x)$ has sort s_2. Axiom (Func Ext) states that two functions f and g of sort $s_1 \ominus s_2$ are equal iff they are behavioral equivalent, i.e., they return the same values on all arguments of sort s_1.

When importing both FUNCTION and TUPLE, we can use $s_1 \otimes \cdots \otimes s_n \ominus s$ to denote the sort for all functions from $s_1, \ldots, s_n$ to s, and write the pattern/axiom $f : s_1 \otimes \cdots \otimes s_n \ominus s$ to specify that f is a multary function from $s_1, \ldots, s_n$ to s.

Multary and nullary functions can be reduced to unary functions. A multary function from sorts $s_1, \ldots, s_n$ to s is a unary function of the function sort $s_1 \otimes \cdots \otimes s_n \ominus s$. A nullary function c of sort s is a unary function from the unit sort *Unit* to s, where *Unit* is a special sort with exactly one element *unit* that is the right identity of application, i.e., $(c\, unit)$ is always equal to c. These are summarized in Table 1.

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

We put everything in the SORT^{++} theory (Fig. 9), which includes a systematic way to represent functions of any arity using a uniform and familiar notation as summarized in Table 1.

The following proposition characterizes the behavior of a function.

```
theory SORT++ Imports: FUNCTION, TUPLE, REL
Symbols: Unit, unit
Notations:
  () ≡ unit
  f(φ) ≡ f φ
  f(φ_1, ..., φ_n) ≡ f ⟨φ_1, ..., φ_n⟩  // for n ≥ 2
Axioms:
  (UNIT SORT)      Unit : Sort
  (UNIT)           unit : Unit
  (UNIT DOMAIN)    ⊤_Unit = unit
  (UNIT IDENTITY)  ∀s : Sort. ∀x : s. (x unit) = x
endtheory
```

Fig. 9. SORT^{++}.

```
theory REL
 Symbols: idRel, converseRel, composeRel
 Notations:
   R^{-1} ≡ converseRel R
   R_1 ∘ R_2 ≡ composeRel R_1 R_2
 Axioms:
   (CONVERSE)     R^{-1} = ∃x. ∃y. ⟨y, x⟩ ∧ (⟨x, y⟩ ∈ R)
   (COMPOSITION)  R_1 ∘ R_2 = ∃x. ∃y. ∃z. ⟨x, z⟩ ∧ (⟨x, y⟩ ∈ R_1 ∧ ⟨y, z⟩ ∈ R_2)
endtheory
```

Fig. 10. REL.

Proposition 4.6. *The following holds for any $n \geq 0$:*

$$\mathsf{SORT}^{++} \vdash \forall s_1, \ldots, s_n : Sort.$$
$$(f : s_1 \otimes \cdots \otimes s_n \ominus s) \to \left(\forall_{1 \leq i \leq n} x_i : s_i .\, f(x_1, \ldots, x_n) : s\right)$$

Remark 4.7. In the rest of the paper, we use the following pattern/axiom to define a function from $s_1, \ldots, s_n$ to s:

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

Following the same idea as in Proposition 4.1, f derives a function $M_f : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ in any given model M that satisfies the axiom (Function). When n is 0, 1, or ≥ 2, M_f is a constant, a unary function, or a multary function of arity n, respectively.

4.4. Relations

In matching logic, a binary relation R can be represented by a pattern that is matched by the pairs of all elements that are in relation R. Since we need to define later various relations, like $\simeq_E$, $\to_E^*$, we introduce the theory REL in Fig. 10, which specifies the inverse of a relation and the composition of two relations. The axioms for the identity relation *idRel* will be added by the importing theory.

5. Capturing order-sorted equational specifications and their initial semantics in matching logic

In this section, we show that the order-sorted equational specification and its initial semantics can be internalized (captured) within matching logic. By "internalization" ("capturing") we mean that the main concepts and reasoning mechanism from order-sorted framework are represented as patterns, theories, axioms, and formal proofs in matching logic. The technical results in this section are obtained by extending the approach in [12].

We first define a matching logic theory that captures:

- the semantics of order-sorted equational specifications: we show that there is a mapping from matching logic models to the corresponding order-sorted algebras; and
- equational (formal) reasoning: formal proofs that are carried out by the equational deduction system can be reproduced by the matching logic proof system.

theory EQSPEC$(S, \leq, F, E)$
Imports: SORT^{++}
Symbols: $s \in S$, $f \in F$, SigOps, $\mathit{SigArgs}$, $\mathit{ConnComp}$
Notations: // $V = \{x_1, \ldots, x_n\}$ is a set of sorted variables
$\forall V\,.\,\varphi \equiv \forall x_1 : \mathsf{sort}(x_1).\ \ldots \forall x_n : \mathsf{sort}(x_n).\,\varphi$
$\exists V\,.\,\varphi \equiv \exists x_1 : \mathsf{sort}(x_1).\ \ldots \exists x_n : \mathsf{sort}(x_n).\,\varphi$
$(s \equiv_{\leq} s') \equiv \langle s, s' \rangle \in \mathit{ConnComp}$
Axioms:

(NonVoid Sorts)	$\top_s \neq \bot$ for $s \in S$
(Sort)	$s : \mathit{Sort}$ for $s \in S$
(SubSort)	$\top_s \subseteq \top_{s'}$ for every $s \leq s'$
(Function)	$f : s_1 \otimes \cdots \otimes s_n \ominus s$ for $f \in F_{s_1 \ldots s_n, s}$
(Signature Ops)	$\top_{\mathit{SigOps}} = \bigvee_{f \in F} f$
(Signature Args)	$\top_{\mathit{SigArgs}} = \bigvee_{f \in F_{s_1 \ldots s_n, s}} \top_{s_1 \otimes \cdots \otimes s_n}$
(ConnComp)	$\mathit{ConnComp} = \mu R.\ \bigvee_{s \leq s'} \vee \exists s : S.\ \langle s, s \rangle \vee R^{-1} \vee (R \circ R)$
(Equation)	$\forall V\,.\,t = t'$ for every $\forall V\,.\,t = t' \in E$

endtheory

Fig. 11. EQSPEC$(S, \leq, F, E)$.

Then, we further extend our matching logic theories of order-sorted algebras to those of the *initial algebras*, and thus capturing the *initial algebra semantics*. The extension is given by capturing the quotient term algebras, which are the typical examples of initial algebras of equational specifications.

5.1. *Capturing order-sorted equational specifications*

Let $(S, \leq, F, E)$ be an order-sorted equational specification. For the sake of presentation we consider only unconditional equations here, but the results can be easily extended to the conditional equations. We add remarks describing what modifications are needed in the matching logic specifications in order to get this extension. In Fig. 11, we show the corresponding matching logic theory. The main components of the theory are listed in the following.

1. Each sort $s \in S$ is declared as a matching logic symbol of sort *Sort*;
2. For each sort s, we add the "nonempty carrier set" axiom $\top_s \neq \bot$, since we are working with nonempty sorts;
3. For each sorts s and s' with $s \leq s'$, we define the subsorting axiom $\top_s \subseteq \top_{s'}$;
4. For each operation $f \in F_{s_1 \ldots s_n, s}$, we define the corresponding axiom $f : s_1 \otimes \cdots \otimes s_n \ominus s$;
5. Equations are wellformed matching logic patterns themselves, thanks to the notations defined in Section 4.1;
6. The equivalence relation $\equiv_{\leq}$ is represented by the symbol *ConnComp* and its corresponding axiom (ConnComp);
7. We define two auxiliary sorts *SigOps* and *SigArgs* for the operations in (S, F) and their argument tuples, respectively; with these sorts, we can quantify over operations and their arguments as $\forall f : \mathit{SigOps}$ and $\forall \mathit{arg} : \mathit{SigArgs}$.

If $E = \emptyset$, then we simply write EQSPEC$(S, \leq, F)$ for EQSPEC$(S, \leq, F, \emptyset)$.

Next, we define the model transformation α that maps any matching logic model of EQSPEC$(S, \leq, F)$ to its corresponding $(S, \leq, F)$-algebra.

Definition 5.1. Let $M \vDash$ EQSPEC$(S, \leq, F)$ be a matching logic model. We define the derived F-algebra $A = \alpha(M)$ as follows:

1. for $s \in S$, the carrier set $A_s = M_s$, where M_s is the inhabitant set of s in M (i.e., the evaluation $|\top_s|_M$).
2. for $f \in F_{s_1 \ldots s_n, s}$, the operation interpretation $A_f = M_f$.

To show that $\alpha(M)$ is a well-defined order-sorted algebra, we need to show that if

$$\text{(a)}\quad f \in F_{s_1 \ldots s_n, s} \cap F_{s'_1 \ldots s'_n, s'},\ s'_i \in [s_i],\ 1 \leq i \leq n,\ s' \in [s]$$

and

$$\text{(b)}\quad (a_1, \ldots, a_n) \in (A_{s_1} \times \cdots \times A_{s_n}) \cap (A_{s'_1} \times \cdots \times A_{s'_n})$$

then

$$A_{f : s_1 \ldots s_n \to s}(a_1, \ldots, a_n) = A_{f : s'_1 \ldots s'_n \to s'}(a_1, \ldots, a_n).$$

However, the above is a direct consequence of the following two axioms: $f : s_1 \otimes \cdots \otimes s_n \ominus s$ and $f : s'_1 \otimes \cdots \otimes s'_n \ominus s'$, which are equivalent to

$$\forall x_1 : s_1, \dots, x_n : s_n . \exists y : s. \, f(x_1, \dots x_n) = y$$

and

$$\forall x'_1 : s_1, \dots, x'_n : s_n . \exists y' : s'. \, f(x'_1, \dots x'_n) = y',$$

where we obtain $x_1 = x'_1, \dots, x_n = x'_n$ imply $f(x_1, \dots x_n) = f(x'_1, \dots x'_n)$ from. Note that no additional constraints on signature are required.

The next result shows that the semantics is properly captured:

Theorem 5.2. *Let $M \vDash \mathsf{EQSPEC}(S, \leq, F, E)$ and $\alpha(M)$ be the derived F-algebra. For any F-equation e, $M \vDash e$ iff $\alpha(M) \vDash_{\mathsf{Alg}} e$. In particular, $\alpha(M)$ is an (F, E)-algebra.*

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

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

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

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

Moreover, the equational reasoning is captured by the matching logic reasoning:

Theorem 5.3. *For any equational specification E and any equation e, the following are equivalent:*

(1) $\mathsf{EQSPEC}(S, \leq, F, E) \vdash e$;
(2) $\mathsf{EQSPEC}(S, \leq, F, E) \vDash e$;
(3) $E \vDash_{\mathsf{Alg}} e$;
(4) $E \vdash_{\mathsf{Alg}} e$.

Proof Sketch. We prove that (1) $\Longrightarrow$ (2) $\Longrightarrow$ (3) $\Longrightarrow$ (4) $\Longrightarrow$ (1). Note that (1) $\Longrightarrow$ (2) is by the soundness of matching logic proof system. (2) $\Longrightarrow$ (3) is by Theorem 5.2. (3) $\Longrightarrow$ (4) is by the completeness of equational deduction. And (4) $\Longrightarrow$ (1) is by the fact that equational deduction can be reproduced by using matching logic proof system. □

Remark 5.4. All results proved in this subsection hold as well if E includes conditional equations and the theory $\mathsf{EQSPEC}(S, \leq, F, E)$ includes an axiom

$$(\textsc{Cond Equation}) \quad \forall V . u_1 = v_1 \wedge \dots \wedge u_n = v_n \to t = t'$$

for every equation$\forall V . u_1 = v_1 \wedge \dots \wedge u_n = v_n \to t = t'$ in E.

5.2. Capturing initial algebra semantics

We define a matching logic theory that captures the term algebra defined by the order-sorted signature, then we refine this theory by adding an axiomatization of the E-equality, for given equations E. The term algebra theory is defined following the famous "no confusion, no junk" slogan.

The theory $\mathsf{NOCONFUSION}(S, F)$ in Fig. 12 has two axioms. (Distinct Function) states that distinct operations are indeed different functions symbols. (No Confusion) defines no-confusion using one pattern.

The matching logic specification for the "no junk" part is split in two theories: $\mathsf{TERM}(S, \leq, F)$ (Fig. 13), which axiomatizes the set of terms, and $\mathsf{NOJUNK}(S, \leq, F)$, which adds to TERM the (No junk) axiom (see Fig. 14). The axiom (Term)

theory NOCONFUSION(S, F)
Axioms:
(Distinct Function) $f \neq f'$ for distinct operation $f, f' \in F$
(No Confusion) $\forall f, f' : SigOps. \forall args, args' : SigArgs.$
$(f\ args) = (f'\ args') \to f = f' \wedge args = args'$
endtheory

Fig. 12. NOCONFUSION(S, F).

theory TERM$(S, \leq, F)$
Symbols: ls, $tinh$
Notations: $[\![s]\!]_F \equiv tinh\ F\ s$
Axioms:
(Least Sort) $\forall t. \exists s : S. (ls\ S\ t) \to s$
$\forall t. \forall s : S. t : s \to (ls\ S\ t) \leq s$
(Preregularity) $\forall s : S. \forall t : s. \lceil ls\ S\ t \rceil$
(Term) $\langle [\![s_1]\!]_F, \ldots, [\![s_n]\!]_F \rangle =$

$$\mu D. \left\langle D_{<s_1} \vee \bigvee_{f \in F_{s_1^1 \ldots s_{m_1}^1, s_1}} f\left(D_{s_1^1}, \ldots, D_{s_{m_1}^1}\right), \ldots, D_{<s_n} \vee \bigvee_{f \in F_{s_1^n \ldots s_{m_n}^n, s_n}} f\left(D_{s_1^n}, \ldots, D_{s_{m_n}^n}\right) \right\rangle$$

where $D_s \equiv (proj\ i\ D)$ s.t. s is equal to s_i, and $D_{<s}$ is $\bigvee_{s' \in S, s' < s} D_{s'}$
endtheory

Fig. 13. TERM$(S, \leq, F)$.

theory NOJUNK$(S, \leq, F)$
Imports: TERM$(S, \leq, F)$
Axioms:
(No junk) $\bigwedge_{s \in S} (\top_s = [\![s]\!]_F)$
endtheory

Fig. 14. NOJUNK$(S, \leq, F)$.

theory TERMALGEBRA$(S, \leq, F)$
Imports: EQSPEC$(S, \leq, F, \emptyset)$, NOJUNK$(S, \leq, F)$, NOCONFUSION(S, F)
Axioms:
(sensibility) $(\bigwedge_{i=1}^n s_i \equiv_\leq s_i') \to s \equiv_\leq s'$ for every $f \in F_{s_1 \ldots s_n, s} \cap F_{s_1' \ldots s_n', s'}$
endtheory

Fig. 15. TERMALGEBRA$(S, \leq, F)$.

in TERM$(S, \leq, F)$ defines the *term carriers* of non-void sorts $[\![s_1]\!]_F, \ldots, [\![s_n]\!]_F$ simultaneously, using one fixpoint pattern $\mu D. \langle \cdots \rangle$, whose ith component $(1 \leq i \leq n)$ is a disjunction over the less sorts and over all operations whose return sorts are s_i applied to the projections of D that correspond to the appropriate argument sorts. Since we work under the pre-regularity assumption, we added axioms for the definition of the least sort and for pre-regularity.

The matching logic theory TERMALGEBRA$(S, \leq, F)$ in Fig. 15 is the aggregation of the theories for the equational specification with no equations, "no confusion", and "no junk" together with the sensibility axiom, needed by Theorem 2.17.

Theorem 5.5. *If $M \vDash$ TERMALGEBRA$(S, \leq, F)$, the derived algebra $\alpha(M)$ is the term algebra T_F, up to isomorphism.*

Proof Sketch. We only need to prove that $\alpha(M)$ satisfies no-junk and no-confusion, where no-confusion is guaranteed by the (No Confusion) axiom in Fig. 12 and no-junk is guaranteed by the (Term) axiom in Fig. 13, which states that the carrier sets of the sorts are the small sets that are closed under the corresponding operations. □

Since $\simeq_E$ is the *smallest relation* that includes the identity relation and all instances of the equations in E, and is closed under converse, composition, and congruence w.r.t. all operations in F, it can be axiomatically defined by the least fixpoint operator μ. This inspires the definition included in the theory EQUIV(S, F, E) (Fig. 16), where we use a symbol *Eq* to

theory EQUIV(S, F, E)
Imports: REL
Symbols: *Eq*, *congRel*
Notations:
$\varphi_1 \subsetneq_E \varphi_2 \equiv \forall x_1 . x_1 \in \varphi_1 \to \exists x_2 . x_2 \in \varphi_2 . \langle x_1, x_2 \rangle \in Eq\, E$
$\varphi_1 \simeq_E \varphi_2 \equiv \varphi_1 \subsetneq_E \varphi_2 \wedge \varphi_2 \subsetneq_E \varphi_1$
Axioms:
(Identity) $idRel = \bigvee_{s \in S} \exists x : s.\ \langle x, x \rangle$
(Congruence) $congRel\ R = \bigvee_{f \in F_{s_1 \ldots s_n, s}} \exists x_1, y_1 : s_1 \ldots \exists x_n, y_n : s_n .\ \langle f(x_1, \ldots, x_n), f(y_1, \ldots, y_n) \rangle \wedge \bigwedge_{1 \le i \le n} \langle x_i, y_i \rangle \in R$
(Equivalence) $Eq\, E = \mu R . idRel \vee R^{-1} \vee (R \circ R) \vee (congRel\ R) \vee \bigvee_{(\forall V . t = t') \in E} \exists V .\ \langle t, t' \rangle$
endtheory

Fig. 16. EQUIV(S, F, E).

capture the congruence relation[1] $\simeq_E$. The theory firstly defines several operations and notations for dealing with (binary) relations. Intuitively, *idRel* is the identity relation on the elements in the carrier sets of sorts in S; R^{-1} and $R_1 \circ R_2$ are the converse of R and composition of R_1 and R_2, respectively and are defined in REL (Fig. 10); *congRel* R is the relation obtained by propagating R through all the operations in F. The last axiom (Equivalence) defines *Eq* as the smallest relation R that includes *idRel*, is closed under converse, composition, and congruence, and includes all equations in E. Finally, we write $\varphi_1 \subsetneq \varphi_2$ to mean that φ_1 is included in φ_2 modulo the relation *Eq*, and $\varphi_1 \simeq \varphi_2$ to mean that φ_1 and φ_2 are the same modulo relation *Eq*.

Let INITIALALGEBRA$(S, \le, F, E)$ be the matching logic specification that imports TERMALGEBRA$(S, \le, F)$, EQUIV(S, F, E).

Theorem 5.6. *For any $M \vDash$ INITIALALGEBRA$(S, \le, F, E)$, the interpretation $(Eq\, E)_M$ forms a binary relation over T_F such that for all ground terms t and t', $(t, t') \in (Eq\, E)_M$ iff $t \simeq_E t'$.*

Note that $\simeq_E$ is also a notation in EQUIV(S, F, E). By Theorem 5.6, this (overloaded) use of notation $\simeq_E$ in EQUIV(S, F, E) is consistent.

Proof. Firstly, note that the equivalence relation $\simeq_E$ in T_F is the smallest relation generated by equational deduction (Definition 2.9). Thus, $\simeq_E$ is defined as the smallest set that includes the identity relation and all (ground) instances of the equations in E, and is closed under symmetry, commutativity, associativity, and congruence.

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

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

Theorem 5.6 naturally leads us to the following key result.

Theorem 5.7. *Under the conditions and notations in Theorem 5.6, we define $\beta(M)$ as the $(Eq\, E)_M$-quotient algebra of $\alpha(M)$. Then, $\beta(M)$ is the quotient term algebra $T_{F/E}$.*

Proof. Since $\alpha(M)$ is the term algebra T_F and Eq_M is the equivalence relation $\simeq_M$, the quotient algebra $\beta(M)$ is by definition the quotient term algebra $T_{F/E}$, according to Theorem 2.16. $\square$

[1] Alternatively, we could have defined *Eq* as a sort inhabited by the pairs in $\simeq_E$ with constructors the equations in E, identities, converse, and composition. This would allow us to use *Eq* in sort constructors (pairs, tuples, functions) similarly to how identity types are used in higher inductive types [25,26]. This is left as future work.

Remark 5.8. All results proved in this subsection hold as well if E includes conditional equations and in the theory $\mathsf{EQUIV}(S, F, E)$ the axiom (Equivalence) is as follows:

$$Eq\, E = \mu R.\, idRel \vee R^{-1} \vee (R \circ R) \vee (congRel\ R) \vee \bigvee_{(\forall V.\, \bigwedge_{i=\overline{1,n}} u_i = v_i \to t = t') \in E} \exists V.\, \langle t, t' \rangle \wedge \bigwedge_{i=\overline{1,n}} \langle u_i, v_i \rangle \in R$$

Recall that the unconditional equations are a particular case ($n = 0$) of the conditional ones.

5.3. Constrained process configurations (CPC) example in matching logic

In this section we exemplify how matching logic captures the initial semantics of the equational order-sorted specifications, specifying a particular case of process configurations used, e.g., in description of mutual exclusion protocols (like QLOCK [4]).

We consider configurations that allows an unbounded number of (numbered) processes that are in one of the three states: “normal” (doing their own things), “waiting” for a resource, and “critical” when using the resource. Such a state is a tuple $\langle n|w|c|q \rangle$ where n, w, c are multisets of identities of the processes that are in “normal”, “waiting”, and “critical” states, respectively, and q is the waiting queue, i.e., an associative list.

5.4. Order-sorted specification of CPC

Here we recall the OSA specification of CPC given in [1], a paper that inspired us in this investigation:

$S = \{Nat, List, MSet, NeMSet, Conf, State, Pred\}$
$\leq = \{Nat < List, Nat < NeMSet < MSet\} \cup =_S$
F_Ω (constructors):
  $\mathbb{0} : \to Nat$, $\mathbb{s}\_ : Nat \to Nat$
  $nil : \to List$, $\_;\_ : List \times List \to List$
  $empty : \to MSet$, $\_\_ : MSet \times MSet \to MSet$,
  $\_\_ : NeMSet \times NeMSet \to NeMSet$
  $\_|\_|\_|\_ : MSet \times MSet \times MSet \times List \to Conf$
  $\langle \_ \rangle : Conf \to State$
  $tt : \to Pred, ff : \to Pred$
$F = \Sigma_\Omega \cup \{dupl : MSet \to Pred, dupl : NeMSet \to Pred\}$
B_Ω:
  associativity for list concatenation $\_\_$
  identity nil for list concatenation $\_\_$
  associativity and commutativity for multiset union $\_;\_$
  identity $empty$ for multiset union $\_;\_$
$B = B_\Omega \cup \emptyset$
$E_\Omega = \emptyset$
$E = E_\Omega \cup \{dupl(s\, u\, u) = tt\}$, where s is any multiset (could be empty).

The corresponding canonical model, denoted $\mathbb{CPC}$, is given as:

$\mathbb{CPC}_{Nat} = \{\mathbb{0}, \mathbb{s}\,\mathbb{0}, \mathbb{s}^2\,\mathbb{0}, \ldots\}$
$\mathbb{CPC}_{List} = \mathbb{CPC}_{Nat} \cup \{nil\} \cup \{n_1; \ldots; n_k \mid n_i \in \mathbb{CPC}_{Nat}, 1 \leq i \leq k, k \geq 2\}$
$\mathbb{CPC}_{NeMSet} = Nat \cup \{[n_1, \ldots, n_k] \mid n_i \in \mathbb{CPC}_{Nat}, 1 \leq i \leq k, k \geq 2\}$
$\mathbb{CPC}_{MSet} = \mathbb{CPC}_{NeMSet} \cup \{empty\}$
$\mathbb{CPC}_{Conf} = \{x_1|x_2|x_3|y \mid x_1, x_2, x_3 \in \mathbb{CPC}_{MSet}, y \in \mathbb{CPC}_{List}\}$
$\mathbb{CPC}_{State} = \{\langle x \rangle \mid x \in \mathbb{CPC}_{Conf}\}$
$\mathbb{CPC}_{Pred} = \{tt, ff\}$

5.5. Matching logic specification of CPC

The matching logic theory CPC in Fig. 17 fully encodes in one page the full initial semantics of the equational specification $(F, E \cup B)$ given in Section 5.4 (together with the corresponding instantiation of the meta-theory in 2.1). The content

theory CPC

Imports: SORT^{++}

Symbols: S, $List$, $MSet$, $NeMSet$, $Conf$, $State$, nil, $conc$, $union$, $conf$, $state$, $dupl$

Notations:

$\varphi_1 \subsetneqq \varphi_2 \equiv \forall x_1 . x_1 \in \varphi_1 \to \exists x_2 . x_2 \in \varphi_2 .\ \langle x_1 , x_2 \rangle \in Eq\, E$

$\varphi_1 \simeq \varphi_2 \equiv \varphi_1 \subsetneqq_E \varphi_2 \wedge \varphi_2 \subsetneqq \varphi_1$

$\exists \overline{x} \equiv \exists x_1 , x_2 , x_3$

$conf\, \overline{x}, y \equiv conf\, x_1\, x_2\, x_3\, y$

$\langle \overline{x}, \overline{y} \rangle \in R \equiv \langle x_1 , x'_1 \rangle \in R \wedge \langle x_2 , x'_2 \rangle \in R \wedge \langle x_3 , x'_3 \rangle \in R$

Axioms:

(Sort Names) $List : Sort \wedge MSet : Sort \wedge NeMSet : Sort \wedge Conf : Sort \wedge State : Sort$

(Sorts Set) $S = Nat \vee List \vee MSet \vee NeMSet \vee Conf \vee State$

(Booleans) $\exists y : Pred .\ y = tt \qquad \exists y : Pred .\ y = ff \qquad \neg (tt \wedge ff) \qquad \top_{Pred} = ff \vee tt$

(Lists-fun) $\forall x, y : List . \exists z : List .\ z = conc\, x\, y \qquad \exists x : List .\ x = nil$

(-dom) $\top_{List} = \mu X . \top_{Nat} \vee nil \vee conc\, X\, X$

(-no-conf.1) $\forall x : Nat .\ nil \neq x \qquad \forall x, y : List .\ nil \neq conc\, x\, y$

(-no-conf.2) $\forall x : Nat . \forall y, z : List .\ x \neq conc\, y\, z$

(-no-conf.3) $\forall x, x', y, y' : List . (conc\, x\, y) \wedge (conc\, x', y') \to (x = x') \wedge (y = y')$

(Msets-fun.1) $\exists y : MSet .\ y = empty \qquad \forall x, y : MSet . \exists z : MSet .\ z = union\, x\, y$

(-cons.2) $\forall x, y : NeMSet . \exists z : NeMSet .\ z = union\, x\, y$

(-dom) $\top_{NeMSet} = \mu X . \top_{Nat} \vee union\, X\, X \qquad \top_{MSet} = empty \vee \top_{NeMSet}$

(-no-conf.1) $\forall x : Nat .\ empty \neq x \qquad \forall x, y : MSet .\ empty \neq union\, x\, y$

(-no-conf.2) $\forall x : Nat . \forall y, z : MSet .\ x \neq union\, y\, z$

(-no-conf.3) $\forall x, x', y, y' : MSet . (union\, x\, y) \wedge (union\, x', y') \to union\, (x \wedge x')\, (y \wedge y')$

(Cfg-fun) $\forall x_1 , x_2 , x_3 : MSet . \forall y : List . \exists z : Conf .\ conf\, x_1\, x_2\, x_3\, y = z$

(-no-conf) $\forall x_1 , x_2 , x_3 , x'_1 , x'_2 , x'_3 : MSet . \forall y, y' : List .\ conf\, x_1\, x_2\, x_3\, y \wedge conf\, x'_1\, x'_2\, x'_3\, y' \to$
$conf\, (x_1 \wedge x'_1)(x_2 \wedge x'_2)(x_3 \wedge x'_3)(y \wedge y')$

(-dom) $\top_{Conf} = conf\, \top_{MSet}\, \top_{MSet}\, \top_{MSet}\, \top_{List}$

(States-fun) $\forall x : Conf . \exists y : State .\ state\, x = y$

(-no-conf) $\forall x, x' : Conf .\ state\, x \wedge state\, x' \to state\, (x \wedge x')$

(-dom) $\top_{State} = state\, \top_{Conf}$

(Dupl-fun) $\forall x : MSet . \exists y : Pred .\ dupl\, x = y$

(-eq.1) $\forall s . \exists s', u .\ s =_{NeMSet} union\, s' (union\, u\, u) \to dupl\, s = tt$

(-eq.2) $\forall s . \forall s', u .\ s \neq_{MSet} union\, s' (union\, u\, u) \to dupl\, s = ff$

(IdRel) $idRel\, R = \exists x : Nat .\ \langle x, x \rangle \wedge \exists x : List .\ \langle x, x \rangle \wedge \exists x : MSet .\ \langle x, x \rangle \wedge$
$\exists x : Conf .\ \langle x, x \rangle \wedge \exists x : State .\ \langle x, x \rangle$

(CongRel) $congRel\, R = (\exists x, y, x' y' : List .\ \langle conc\, x\, y, conc\, x'\ y' \rangle \wedge \langle x, x' \rangle \in R \wedge \langle y, y' \rangle \in R) \vee$
$(\exists x, y, x' y' : MSet .\ \langle union\, x\, y, union\, x'\ y' \rangle \wedge \langle x, x' \rangle \in R \wedge \langle y, y' \rangle \in R) \vee$
$(\exists \overline{x}, \overline{x'} : MSet . \exists y : List .\ \left\langle conf\, \overline{x}\, y, conf\, \overline{x'}\ y' \right\rangle \wedge \left\langle \overline{x}, \overline{x'} \right\rangle \in R \wedge \langle y, y' \rangle \in R) \vee$
$(\exists x, x' : Conf .\ \langle state\, x, state\, x' \rangle \wedge \langle x, x' \rangle \in R)$

(Equiv) $Eq = \mu R .\ idRel \vee R^{-1} \vee (R \circ R) \vee$
$(congRel\ \ R) \vee (\exists x, y, z : List .\ \langle conc (conc\, x\, y)\, z, conc\, x\, (conc\, y\, z) \rangle) \vee$
$(\exists x : List .\ \langle conc\, x\, nil, x \rangle \vee \langle conc\, nil\, x, x \rangle) \vee$
$(\exists x, y, z : MSet .\ \langle union (union\, x\, y)\, z, union\, x\, (union\, y\, z) \rangle) \vee$
$(\exists x : MSet .\ \langle union\, x\, empty, x \rangle) \vee (\exists x, y : MSet .\ \langle union\, x\, y, union\, y\, x \rangle)$

endtheory

Fig. 17. CPC.

of the theory is self-explained. In order to keep it compact, there are several places with multiple axioms on the same line (e.g., (Booleans)). Recall that the specification of the natural numbers (NAT) is included in SORT^{++}. For each data type, there are axioms declaring the constructors as being functional ((-fun)), the "no junk" axiom(s) ((-dom)), and the "no-confusion" axioms ((-no-conf)). The equations $B = B_\Omega$ are captured by $\simeq$. The only equation in E is encoded by (-eq.1) and (-eq.2). We used matching-logic notation, but we may use sugar syntax to give a more domain specific flavor:

Notations:

$x ; y \equiv conc\, x\, y$

$x_1 | x_2 | x_3 | y \equiv conf\, x_1\, x_2\, x_3\, y$

However, the use of __ for multisets union could be a bit confusing since the same notation is used for matching logic application.

The partial order $\leq$ over sorts is not specified since it can be deduced from the domain axioms. For instance, from $\top_{List} = \mu X . \top_{Nat} \vee nil \vee conc\, X\, X$ we may deduce $Nat \leq List$, and from $\top_{MSet} = empty \vee \top_{NeMSet}$ we may deduce $NeMSet \leq MSet$. The axioms for pre-regularity and sensibility are also not included since they can be deduced from the specification. The well-definedness of *dupl* follows by showing that CPC $\vdash \top_{Pred} = \top_{Pred} \vee dupl\, \top_{NeMSet}$. Moreover, the axioms in CPC are not independent. For instance, the functional axiom $\forall x, y : NeMSet. \exists z : NeMSet. z = union\, x\, y$ can be deduced from the other ones.

The proofs (of some) of the above properties require inductive reasoning. Matching logic proof system includes Knaster-Tarski proof rule [8,22] dedicated to fixpoint reasoning. This rule, combined with the least fixpoint definitions leads to matching logic proof strategies (see [22] for more details on (co)inductive reasoning in matching logic). Here we exemplify the inductive strategies for natural numbers, lists and multisets.

We may use matching logic proof system to reason on CPC properties. E.g., the following propositions show that the inductive reasoning for the used data types is obtained for free due to Knaster-Tarski proof rule and the NoJunk axiom.

Proposition 5.9 *(Peano Induction).*

$$\mathsf{CPC} \vdash (\mathbb{0} \in N \wedge \mathbb{s}\, N \to N) \to \forall x : Nat. x \in N$$

The Peano induction principle stated by Proposition 5.9 is a bit non-standard. A more familiar formulation is $\varphi(\mathbb{0}) \wedge (\forall y : Nat. \varphi(y) \to \varphi(\mathbb{s}\, y)) \to \forall x : Nat. \varphi(x)$. We show that two formulations are equivalent. In Proposition 5.9, N is a set variable. If we take $N \equiv \exists x. x \wedge \varphi$, then $\mathbb{0} \in N$ is equivalent to $\varphi(\mathbb{0})$, $\mathbb{s}\, N \to N$ is equivalent to $\forall y : Nat. \varphi(y) \to \varphi(\mathbb{s}\, y)$, and $x \in N$ is equivalent to $\varphi(x)$ holds.

Since the specifications for lists and multisets include axioms over constructors B_Ω, the induction principles for them must take into account the equivalence $\simeq$.

Proposition 5.10 *(List Induction).*

$$\mathsf{CPC} \vdash ((\forall \ell. \ell \simeq nil \to \ell \in L) \wedge (\forall x : Nat. x \in L) \wedge (conc\, L\, L) \subsetneq L) \to \forall \ell : List. \ell \in L$$

The principle stated by Proposition 5.10 is equivalent to

$$\begin{aligned}&(\varphi(nil) \wedge (\forall x : Nat. \varphi(x)) \wedge (\forall \ell_1, \ell_2 : List. \varphi(\ell_1) \wedge \varphi(\ell_2) \to \varphi(conc\, \ell_1\, \ell_2)) \wedge \\ &\ \forall \ell_1, \ell_2 : List. \ell_1 \simeq \ell_2 \to (\varphi(\ell_1) \leftrightarrow \varphi(\ell_2)) \qquad . \\ &) \to \forall \ell : List. \varphi(\ell)\end{aligned}$$

Proposition 5.11 *(Multiset Induction).*

$$\begin{aligned}&\mathsf{CPC} \vdash ((\forall x : Nat. x \in M) \wedge (union\, M\, M) \subsetneq M) \to \forall m : NeMSet. m \in M \\ &\mathsf{CPC} \vdash ((\forall m. m \simeq empty \to m \in M) \wedge (\forall x : Nat. x \in M) \wedge (union\, M\, M) \subsetneq M) \to \\ &\qquad \forall m : MSet. m \in M\end{aligned}$$

The principle stated by Proposition 5.10 is equivalent to

$$\begin{aligned}&((\forall x : Nat. \varphi(x)) \wedge (\forall m_1, m_2 : NeMSet. \varphi(m_1) \wedge \varphi(m_2) \to \varphi(union\, m_1\, m_2)) \wedge \\ &\ \forall m_1, m_2 : List. m_1 \simeq m_2 \to (\varphi(m_1) \leftrightarrow \varphi(m_2)) \\ &) \to \forall m : NeMSet. \varphi(m)\end{aligned}$$

and

$$\begin{aligned}&(\varphi(empty) \wedge (\forall x : Nat. \varphi(x)) \wedge (\forall m_1, m_2 : MSet. \varphi(m_1) \wedge \varphi(m_2) \to \varphi(union\, m_1\, m_2)) \wedge \\ &\ \forall m_1, m_2 : List. m_1 \simeq m_2 \to (\varphi(m_1) \leftrightarrow \varphi(m_2)) \\ &) \to \forall m : MSet. \varphi(m)\end{aligned}$$

6. Capturing constructor decomposition of order-sorted equational theories

The definition of the constructor decomposition of order-sorted equational theories is based on convergent rewriting modulo a set of axioms (unconditional equations), and protecting specification inclusion. We firstly show that all these notions can be fully axiomatized in matching logic.

6.1. Capturing rewriting relations given by conditional equations

Recall that the rewrite rules in $\vec{E}$ are obtained by orienting the conditional equations E. Since the matching logic specification of $\vec{E}$-rewriting modulo B is not quite trivial, we give a detailed definition for it here.

We start with an example to understand the challenges we have to address. Assume that E includes the following equations over natural numbers and list of natural numbers:

$$\begin{array}{ll} gt\,(\mathbb{s}\,m)\,\mathbb{0} = tt & gt\,\mathbb{0}\,(\mathbb{s}\,n) = ff \\ gt\,n\;n = ff & gt\,(\mathbb{s}\,m)\,(\mathbb{s}\,n) = gt\,m\,n \\ (gt\,m\,n = tt) \to m;n = n;m & \text{(conditional equation)} \end{array}$$

and that B includes associativity for ;. Then $\vec{E}$ is:

$$\begin{array}{ll} gt\,(\mathbb{s}\,m)\,\mathbb{0} \Rightarrow tt & gt\,\mathbb{0}\,(\mathbb{s}\,n) \Rightarrow ff \\ gt\,n\,n \Rightarrow ff & gt\,(\mathbb{s}\,m)\,(\mathbb{s}\,n) \Rightarrow gt\,m\,n \\ gt\,m\,n = tt \to m;n \Rightarrow n;m & \text{(conditional rule)} \end{array}$$

The rewrite system is obviously convergent modulo B. A first challenge is given by the fact that definition for one-step rewriting and rewriting (with an arbitrary number of steps) are mutually recursive: the definition of one-step rewriting requires the definition of rewriting, and the definition of general rewriting requires that of one-step rewriting. For instance, $\mathbb{s}\,\mathbb{s}\,\mathbb{0};\mathbb{s}\,\mathbb{0} \Rightarrow \mathbb{s}\,\mathbb{0};\mathbb{s}\,\mathbb{s}\,\mathbb{0}$ is one-step rewriting, even if its condition $gt\,(\mathbb{s}\,\mathbb{s}\,\mathbb{0})(\mathbb{s}\,\mathbb{0}) = tt$ is obtained by a two-step rewriting: $gt\,(\mathbb{s}\,\mathbb{s}\,\mathbb{0})(\mathbb{s}\,\mathbb{0}) \Rightarrow gt\,(\mathbb{s}\,\mathbb{0})\mathbb{0} \Rightarrow tt$.

We assume that the equations in E are of the form

$$\bigwedge_{i\in I} u_i = v_i \to t = t',$$

where I is a finite set. If $I = \emptyset$ then we have an unconditional equation $t = t'$. We assume that $\vec{E}$ includes the conditional rewrite rules

$$\bigwedge_{i\in I} u_i =_{\vec{E}\downarrow B} v_i \to t \Rightarrow t',$$

where $u =_{\vec{E}\downarrow B} v$ holds iff there are $\vec{E}$-irreducible terms $\bar{u}$ and $\bar{v}$ such that $u \Rightarrow^* \bar{u}$, $v \Rightarrow^* \bar{v}$, and $\bar{u} =_B \bar{v}$. When $\vec{E}$ is convergent modulo B, we may take $\bar{u} = u!_{\vec{E},B}$, $\bar{v} = v!_{\vec{E},B}$, and $=_{\vec{E}\downarrow B}$ coincides with $=_{E\cup B}$.

The matching logic specification of the rewriting relations is given in Fig. 18. The pattern $rew\,\vec{E}\,B$ matches all pairs $\langle u, u'\rangle$ such that $u \Rightarrow^*_{\vec{E},B} u'$ (u $\vec{E}$-rewrites modulo B into u'). The axiom (FINITE-REWRITING) is mainly based on the deduction rules for the conditional rewriting logic [27]: $Eq\,B$ gives the idle rewrites modulo B, $\{u \Rightarrow u' \mid \langle u, u'\rangle \in Eq\,B\}$, $R \circ R$ gives the transitivity, $congRel\,R$ the congruence, and the last part the nested replacement.

The pattern $rew^1\,\vec{E}\,B$ matches all one-step $\vec{E}$-rewrites modulo B and its definition given by (ONE-STEP REWRITING) is based on (ONE-STEP CONGRUENCE), which constraints rewriting to only one-step rewriting of a single argument. Note the use of finite rewriting in conditions. Since the greatest fixpoint operator ν is used in the axiom ((IN)FINITE-REWRITING), the pattern $rew^\infty\,\vec{E}\,B$ matches the set of finite and infinite (if any) rewrites.

Proposition 6.1. *The following internal theorems show that the rewriting relations defined by* $\mathsf{REW}(S, F, E, B)$ *are consistent:*

1. $\mathsf{REW}(S, F, E, B) \vdash rew^1\,\vec{E}\,B \subseteq rew\,\vec{E}\,B$
2. $\mathsf{REW}(S, F, E, B) \vdash rew\,\vec{E}\,B = \mu R.\,Eq\,B \vee rew^1\,\vec{E}\,B \vee R \circ R$
3. $\mathsf{REW}(S, F, E, B) \vdash rew\,\vec{E}\,B \subseteq rew^\infty\,\vec{E}\,B$

The proof of Proposition 6.1 follows direct from definitions and (co)inductive reasoning.

6.2. Capturing convergent rewriting modulo B

If $\vec{E}$ is convergent modulo B then $\vec{E}$-rewriting modulo B can be used as a decision procedure for $E \cup B$-equality: $u =_{E\cup B} v$ iff $u!_{\vec{E},B} =_B v!_{\vec{E},B}$, provided that $=_B$ is decidable. The specification $\mathsf{CONVREW}(S, F, E, B)$ supplies a matching logic axiomatizations of the properties that defines the convergence modulo B (see Fig. 19). Since $\vec{E}$ is convergent modulo B now, we may improve the specification by replacing the notation for $u =_{\vec{E}\downarrow B} v$ by $u =_{\vec{E}\downarrow B} v \equiv \langle u!_{rew\,\vec{E}\,B}, v!_{rew\,\vec{E}\,B}\rangle \in Eq\,B$.

theory REW(S, F, E, B)
Imports: EQUIV(S, F, B)
Symbols: $ired$, $congRel^1$, rew, rew^1, rew^∞
Notations:

$ired\,t\,R\,B \equiv \forall t'.\, \langle t, t'\rangle \in R \to \langle t, t'\rangle \in Eq\,B$
$t!_R \equiv \exists \bar{t}.\, \bar{t} \wedge \langle t, \bar{t}\rangle \in R \wedge ired\,\bar{t}\,R\,B$
$u =_{\vec{E}\downarrow B} v \equiv \exists \bar{u}, \bar{v}.\, \langle u, \bar{u}\rangle \in rew\,\vec{E}\,B \wedge \langle v, \bar{v}\rangle \in rew\,\vec{E}\,B \wedge \langle \bar{u}, \bar{v}\rangle \in Eq\,B$

Axioms:

(FINITE REWRITING)

$$rew\,\vec{E}\,B = \mu R.\, Eq\,B \vee R \circ R \vee congRel\;R \vee \left(\bigvee_{\forall V.\, \bigwedge_{i\in I} u_i = v_i \to t = t' \in E} \exists V.\, \exists u, u'.\, \langle u, u'\rangle \wedge \langle u, t\rangle \in Eq\,B \wedge \langle t', u'\rangle \in Eq\,B \wedge \forall i : I.\, \exists u'_i, v'_i.\, \langle u_i, u'_i\rangle \in R \wedge \langle v_i, v'_i\rangle \in R \wedge \langle u'_i, v'_i\rangle \in Eq\,B \right)$$

(ONE-STEP CONGRUENCE)

$$congRel^1\,B\,R = \bigvee_{f\in F_{s_1\ldots s_n,s}} \exists x_1, y_1 : s_1 \ldots \exists x_n, y_n : s_n.\, \langle f(x_1, \ldots, x_n), f(y_1, \ldots, y_n)\rangle \wedge \exists j.\, 1 \le j \le n \wedge \langle x_j, y_j\rangle \in R \wedge \bigwedge_{i=\overline{1,n}, i\neq j} \langle x_i, y_i\rangle \in Eq\,B$$

(ONE-STEP REWRITING)

$$rew^1\,\vec{E}\,B = \mu R.\, congRel^1\,B\,R \vee \left(\bigvee_{\forall V.\, \bigwedge_{i\in I} u_i = v_i \to t = t' \in E} \exists V.\, \exists u, u'.\, \langle u, u'\rangle \wedge \langle u, t\rangle \in Eq\,B \wedge \langle t', u'\rangle \in Eq\,B \wedge \forall i : I.\, u_i =_{\vec{E}\downarrow B} v_i \right)$$

((IN)FINITE REWRITING)

$rew^\infty\,\vec{E}\,B = \nu R.\, Eq\,B \vee rew^1\,\vec{E}\,B \vee R \circ R$
endtheory

Fig. 18. REW(S, F, E, B).

theory CONVREW(S, F, E, B)
Imports: REW(S, F, E, B)
Axioms:

(SORT DECREASING) $\forall t, t'.\, \langle t, t'\rangle \in (rew\,\vec{E}\,B) \to (ls\,S\,t') \le (ls\,S\,t)$
(STRICT COHERENCE) $\forall t_1, t_2, t'_1.\, \langle t_1, t'_1\rangle \in Eq\,B \wedge \langle t_1, t_2\rangle \in rew\,\vec{E},\ B \to$
$\exists t'_2.\, \langle t_2, t'_2\rangle \in Eq\,B \wedge \langle t'_1, t'_2\rangle \in rew\,\vec{E},\ B$
(CONFLUENCE) $\forall t, t_1, t_2.\, (\langle t, t_1\rangle \in (rew\,\vec{E}\,B) \wedge \langle t, t_2\rangle \in (rew\,\vec{E}\,B)) \to$
$\exists t'.\, (\langle t_1, t'\rangle \in (rew\,\vec{E}\,B) \wedge \langle t_2, t'\rangle \in (rew\,\vec{E}\,B))$
(TERMINATING) $(rew\,\vec{E}\,B) = (rew^\infty\,\vec{E}\,B)$
endtheory

Fig. 19. CONVREW(S, F, E, B).

Proposition 6.2. *The following internal theorems show the main properties of the convergent rewriting:*

1. CONVREW$(S, F, E, B) \vdash \forall t.\, \lceil t!_{rew\,\vec{E}\,B} \rceil$
2. CONVREW$(S, F, E, B) \vdash \forall t, u, v.\, (u = t!_{rew\,\vec{E}\,B} \wedge v = t!_{rew\,\vec{E}\,B}) \to \langle u, v\rangle \in Eq\,B$
3. CONVREW$(S, F, E, B) \vdash \forall u, v.\, \langle u, v\rangle \in Eq\,E \cup B \leftrightarrow \langle u!_{rew\,\vec{E}\,B}, v!_{rew\,\vec{E}\,B}\rangle \in Eq\,B$

The proof of Proposition 6.2 follows direct from definitions and inductive reasoning only (since the greatest fixpoint operator is no longer used, due to (TERMINATING) axiom).

6.3. *Capturing constructor decomposition of order-sorted equational theories*

We consider specifications $(S, \le, F, E \cup B)$ satisfying condition I1 given in Section 2, and sub-signatures $(\Omega, E_\Omega \cup B_\Omega)$ with $\Omega \subseteq F$, $E_\Omega \subseteq E$, and $B_\Omega \subseteq B$. $(\Omega, E_\Omega \cup B_\Omega)$ is a sub-signature of constructors if it satisfies the following constraints: B-pre-regularity, to be imported in a protected way, and sufficient completeness. All these constraints are axiomatized by the specification CONS$(S, F, E, B, \Omega, E_\Omega, B_\Omega)$ given in Fig. 20. Obviously, $\Omega \subseteq F$ implies NOCONFUSION$(S, F) \models$ NOCONFUSION(S, Ω) and EQSPEC$(S, \le, F) \models$ EQSPEC$(S, \le, \Omega)$.

theory $\mathsf{CONS}(S, F, E, B, \Omega, E_\Omega, B_\Omega)$
Imports: $\mathsf{TERMALGEBRA}(S, \leq, F)$, $\mathsf{TERM}(S, \leq, \Omega)$, $\mathsf{EQUIV}(S, \leq, F, E_\Omega \cup B_\Omega)$
$\mathsf{CONVREW}(S, F, E, B)$, $\mathsf{CONVREW}(S, \Omega, E_\Omega, B_\Omega)$
Axioms:
(B-PREREGULARITY)
$\forall v, v'. \langle v, v' \rangle \in \bigvee_{u=u' \in B} \exists var(u). \langle u, u' \rangle \to (ls\, S\, v) = (ls\, S\, v')$
(PROTECT I)
$\forall s, t, t'. s \in S \wedge t \in [\![s]\!]_\Omega \wedge t' \in [\![s]\!]_\Omega \to (\langle t, t' \rangle \in Eq\, B_\Omega \leftrightarrow \langle t, t' \rangle \in Eq\, B)$
(PROTECT II)
$\forall s, t. s \in S \wedge t \in [\![s]\!]_\Omega \to (ired\, t\, (rew\, E_\Omega\, B_\Omega)\, B_\Omega) = (ired\, t\, (rew\, E\, B)\, B)$
(PROTECT III)
$\forall s. s \in S \to (\exists t : s. t!_{rew\, \vec{E}_\Omega\, B_\Omega}) = (\exists t : s. t!_{rew\, \vec{E}\, B})$
(SUFFICIENT COMPLETENESS I)
$\forall s, t. s \in S \wedge t \in [\![s]\!]_F \to \exists s'. s' \in S \wedge t!_{rew\, \vec{E}\, B} \subseteq [\![s']\!]_\Omega$
(SUFFICIENT COMPLETENESS II)
$\forall s, u, v. s \in S \wedge (u \in [\![s]\!]_\Omega \wedge \langle u, v \rangle \in Eq\, B) \to v \in [\![s]\!]_\Omega$
endtheory

Fig. 20. $\mathsf{CONS}(S, F, E, B, \Omega, E_\Omega, B_\Omega)$.

The next definitions show how the models involved in the definition of constructor decomposition are obtained, up to isomorphism, from a $\mathsf{CONS}(S, \leq, F, E, B, \Omega, E_\Omega, B_\Omega)$-model.

Definition 6.3. Let $M \vDash \mathsf{CONS}(S, \leq, F, E, B, \Omega, E_\Omega, B_\Omega)$ be a matching logic model.
The derived F-algebra $A = \alpha_\Lambda(M)$, $\Lambda \in \{F, \Omega\}$, is defined as below:

- for $s \in S$, the carrier set $A_s = \left| [\![s]\!]_\Lambda \right|_M$;
- for $f \in \Lambda_{s_1 \ldots s_n, s}$, the operation interpretation $A_f = M_f|_A$.

Definition 6.4. Assume the notations in Definition 6.3. The derived F-algebra $\beta_\Lambda(M)$ is the $(Eq\, E \cup B)$-quotient algebra of $\alpha_\Lambda(M)$.

Definition 6.5. Let $M \vDash \mathsf{CONS}(S, F, E, B, \Omega, E_\Omega, B_\Omega)$ be a matching logic model.
The derived F-algebra $A! = \alpha!_\Lambda(M)$, $\Lambda \in \{F, \Omega\}$, is defined as below:

- for $s \in S$, the carrier set $A!_s = \{ \left| t!_{rew\, \vec{E_\Lambda}\, B} \right|_M \mid t \in [\![s]\!]_\Lambda \}$, where $E_F = E$;
- for $f \in F_{s_1 \ldots s_n, s}$, the operation interpretation $A!_f$ is defined by $A!_f(|t!_1|_M, \ldots, |t!_n|_M) = |f(t!_1, \ldots, t!_n)!|_M$.

Definition 6.6. Assume the notations in Definition 6.5. The derived F-algebra $\gamma_\Lambda(M)$ is the $(Eq\, B_\Lambda)$-quotient algebra of $\alpha!_\Lambda(M)$, where $B_F = B$.

The next theorem shows that $M \models \mathsf{CONS}(S, F, E, B, \Omega, E_\Omega, B_\Omega)$ captures indeed exactly the models required by the constructor constrained patterns.

Theorem 6.7. *If $M \models \mathsf{CONS}(S, F, E, B, \Omega, E_\Omega, B_\Omega)$ then:*

1. *the algebra $\beta_F(M)$ is the quotient term algebra $T_{F/E \cup B}$ up to isomorphism;*
2. *the algebra $\beta_\Omega(M)$ is the quotient term algebra $T_{\Omega/E_\Omega \cup B_\Omega}$ up to isomorphism;*
3. *the algebra $\gamma_F(M)$ is the canonical algebra $C_{F/E,B}$ up to isomorphism;*
4. *the algebra $\gamma_\Omega(M)$ is the canonical algebra $C_{\Omega/E_\Omega, B_\Omega}$ up to isomorphism;*
5. *$M \models \forall s. s \in S \to [\![s]\!]_\Omega \simeq_{E \cup B} [\![s]\!]_F$.*

Proof Sketch. 1. Since $\mathsf{CONS}(S, \leq, F, \Omega, E, B, E_\Omega, B_\Omega)$ includes $\mathsf{TERMALGEBRA}(S, \leq, F)$ and $\mathsf{EQUIV}(S, \leq, F, E \cup B)$, it follows that

$$\mathsf{CONS}(S, \leq, F, \Omega, E, B, E_\Omega, B_\Omega) \models \mathsf{INITIALALGEBRA}(S, \leq, F, E \cup B).$$

Then we obtain that $\beta_F(M)$ is the quotient term algebra $T_{F/E \cup B}$ (up to isomorphism) by Theorem 5.7. Recall that by Remark 5.8, this theorem holds for conditional equations E as well.

2. We obtain that $\alpha_\Omega(M)$ isomorphic to T_Ω similarly to Theorem 5.5 (the proof of this theorem does not use the no-junk axiom, but only those defining terms). The rest of the proof is similar to that of Theorem 5.6.
3. The algebra $\alpha!_F(M)$ is well defined due to the axioms from $\mathsf{CONVREW}(S, F, E, B)$ and it is isomorphic to the algebra of the canonical forms C_F. The rest of the proof is similar to that of Theorem 5.6, but using now Proposition 6.2.
4. Similar to 3.
5. It is a consequence of the axioms specifying the sufficient completeness. □

The proof of the next result follows directly from the definition of the specification and the above theorems.

Theorem 6.8. *The requirements I2-I7 from Section 2.2 can be derived from* $\mathsf{CONS}(S, \leq, F, \Omega, E, B, E_\Omega, B_\Omega)$.

7. Application to constrained constructor patterns

Reachability logic [28,29] is a *logic scheme* for specifying and reasoning about transition systems, where formulas, called *reachability rules*, are used to express reachability properties, which state that the target states can be reached from the source states, following the transitions of the system. A reachability logic consists of:

- A specification language for states.
- A specification language for one-step transitions.
- A set of reachability formulas of the form $\Phi \longrightarrow^{\circledast} \Phi'$, which state that any maximal transition starting from a state satisfying Φ "must" reach a state satisfying Φ'.

There are (at least) three interpretations for "must". Firstly, it can be *partial*: Φ' is required to be satisfied only for maximal transitions that are *finite*. Secondly, it can be *total*: Φ' must be satisfied by all (finite or infinite) maximal transition sequences. Thirdly, it can be *invariant*: Φ' must be satisfied by all states reachable from Φ. All these kinds of reachability properties can be naturally expressed in matching logic [8].

Reachability logic for rewrite theories [1,2] is a reachability logic, where

- the state specifications are given by *constrained constructor patterns*,
- the one-step transition specifications are given by a rewrite theory.

We assume a $(S, \leq, \Omega, E_\Omega \cup B_\Omega) \subseteq (S, \leq, F, E \cup B)$ a constructor decomposition of order-sorted equational theories.

Definition 7.1. Given $s \in S$, an *s-sorted atomic constrained constructor pattern predicate* is an expression $u|\varphi$, where $u \in T_\Omega(X)$ has sort s and φ is a quantifier-free $(F \cup \{=\})$-formula; see [2, pp. 204]. The set of *constrained constructor pattern predicates* $PatPred(\Omega, \Sigma)$, is the smallest set that includes $\perp$ and atomic constrained constructor pattern predicates, and is closed under disjunction and conjunction. The *semantics* of a constrained constructor pattern predicate Φ is the set $[\![\Phi]\!]_C$ of canonical terms that *satisfies* it:

- $[\![\perp]\!]_C = \emptyset$.
- $[\![\Phi \vee \Phi']\!]_C = [\![\Phi]\!]_C \cup [\![\Phi']\!]_C$.
- $[\![\Phi \wedge \Phi']\!]_C = [\![\Phi]\!]_C \cap [\![\Phi']\!]_C$.
- $[\![u|\varphi]\!]_C = \{canf(\rho(u)) \mid \rho\colon X \to T_\Omega, C_{F/E,B} \models \rho(\varphi)\}$.

An example of an atomic constrained constructor pattern predicate for CPC example is $\langle n|w|c|q\rangle|dupl(n\,w\,c) \neq tt$, since no process can be waiting and critical at the same time.

There are several additional operations over constrained constructor patterns required to express reachability properties and to support their verification in a computational efficient way. These include (parameterized) subsumption, over-approximation of the complements, and parameterized intersections. The definitions of these operations exploit the cases when the matching and unification modulo $E \cup B$ can be efficiently solved, using, e.g., the theory of variants [5,6]. In the following, we briefly recall their definitions.

Subsumption The following question is often met during the process of proving reachability properties: When is the constrained constructor pattern $u|\psi$ an instance of a finite family $\{(v_i|\psi_i) \mid i \in I\}$, i.e., $[\![u|\varphi]\!] \subseteq \bigcup_{i\in I}[\![v_i|\psi_i]\!]$? The answer is given by $E_\Omega \cup B_\Omega$-matching. Let $\textsc{match}(u, \{v_i \mid i \in I\})$ denote a complete set of (parameter-preserving) set of pairs (i, β) with β a substitution such that $u =_{E_\Omega \cup B_\Omega} v_i\beta$, i.e., β matches v_i on u modulo $E_\Omega \cup B_\Omega$. Assuming that $u|\psi$ and $\{(v_i|\psi_i) \mid i \in I\}$ do not share variables, the constrained constructor pattern subsumption is formally defined as follows:

Definition 7.2 *([1,2]).* A family of constrained constructor patterns $\{(v_i|\psi_i) \mid i \in I\}$ *subsumes* $u|\varphi$, denoted $u|\varphi \sqsubseteq \{(v_i|\psi_i) \mid i \in I\}$, iff

$$C_{\Sigma/B,E} \models \varphi \rightarrow \bigvee_{(i,\beta)\in\text{MATCH}(u,\{v_i|i\in I\})} \psi_i\beta$$

Over-approximation of the complements In [1,2] it is showed that the complement of a constrained constructor pattern cannot be computed using negation, i.e., $[\![u|\top]\!] \setminus [\![u|\varphi]\!] = [\![u|\neg\varphi]\!]$ does not always hold, but the inclusion $[\![u|\top]\!] \setminus [\![u|\varphi]\!] \subseteq [\![u|\neg\varphi]\!]$ holds. Therefore,

Definition 7.3. An *over-approximation* is defined as:

$$[\![u|\varphi]\!] \setminus\!\!\setminus [\![u|\psi]\!] \;\triangleq\; [\![u|\varphi]\!] \cap [\![u|\neg\psi]\!] \qquad (\text{equals to } [\![u|\varphi \wedge \neg\psi]\!])$$

Parameterized intersections The intersection of two constrained constructor patterns that share a set of variables Y is defined as

$$(u|\varphi) \wedge_Y (v|\psi) \triangleq \bigvee_{\alpha\in Unif_{E_\Omega\cup B_\Omega}(u,v)} (u|\varphi \wedge \psi\alpha)$$

where $Unif_{E_\Omega\cup B_\Omega}(u,v)$ is a complete set of $E_\Omega \cup B_\Omega$-unifiers (the parameterized intersection is defined only when such a set exists). We have

$$[\![(u|\varphi) \wedge_Y (v|\psi)]\!] = \bigcup_{\rho\in[Y\rightarrow T_\Omega]} [\![(u|\varphi)\rho]\!] \cap [\![(v|\psi)\rho]\!]$$

Parameterized containment Given the constrained constructor patterns $u|\varphi$ and $\{(v_i|\psi_i) \mid i \in I\}$ with the shared variables Z, their set containment is defined as follows:

$$[\![u|\varphi]\!] \subseteq_Z [\![\bigvee_{i\in I}(v_i|\psi_i)]\!] \quad \text{iff} \quad \forall\rho \in [Z \rightarrow T_\Omega] \text{ s.t. } [\![(u|\varphi)\rho]\!] \subseteq [\![\bigvee_{i\in I}(v_i|\psi_i)\rho]\!]$$

The Z*-parameterized subsumption* of $u|\varphi$ by $\{(v_i|\psi_i) \mid i \in I\}$, written

$$u|\varphi \sqsubseteq_Z \bigvee_{i\in I}(v_i|\psi_i)$$

holds, iff $C_{\Sigma/E,B} \models \varphi \rightarrow \bigvee_{(i,\beta)\in\text{MATCH}(u,\{v_i|i\in I\},Z)}(\psi_i\beta)$. We have $u|\varphi \sqsubseteq_Z \bigvee_{i\in I}(v_i|\psi_i)$ implies $[\![u|\varphi]\!] \subseteq_Z [\![\bigvee_{i\in I}(v_i|\psi_i)]\!]$.

7.1. Capturing constrained constructor patterns in matching logic

The next result is a direct consequence of Theorem 6.7.

Theorem 7.4. *Let $M \models \text{CONS}(S, \leq, F, \Omega, E, B, E_\Omega, B_\Omega)$ be a matching logic model. If $u|\varphi$ is a constrained constructor pattern, then $[\![u|\varphi]\!]_C = \big|[\![u \wedge \varphi]\!]_\Omega\big|_M$.*

Next we explain in matching logic terms some of the constrained constructor pattern operations discussed in the previous section. We regard a substitution $\sigma \triangleq \{x_1 \mapsto t_1, \ldots, x_n \rightarrow t_n\}$ as the matching logic pattern $\sigma^{\mathsf{ML}} \triangleq x_1 = t_1 \wedge \cdots \wedge x_n = t_n$.

7.1.1. Subsumption

Let us discuss the matching logic counterpart of the subsumption. The matching logic pattern that corresponds to $[\![u|\varphi]\!] \subseteq \bigcup_{i\in I}[\![(v_i|\psi_i)]\!]$, is the following:

$$\left(\exists\overline{x}:\overline{s}.\, u \wedge \varphi\right) \subseteq \left(\bigvee_{i\in I} \exists\overline{y_i}:\overline{s_i}.\, v_i \wedge \psi_i\right)$$

where $\overline{x}:\overline{s} = FV(u|\varphi)$, and $\overline{y_i}:\overline{s_i} = FV(v_i|\psi_i)$. Since the two patterns do not share variables by assumption, the above is a well-formed matching logic pattern (we remind that $\varphi \subseteq \varphi'$ is the sugar-syntax of the ML pattern $\lfloor\varphi \rightarrow \varphi'\rfloor$).

The matching logic translation of the definition for $u|\varphi \sqsubseteq \{(v_i|\psi_i) \mid i \in I\}$ is

$$\varphi \rightarrow \bigvee_{(i,\beta)\in\text{MATCH}(u,\{v_i|i\in I\})} \left(\psi_i \wedge \beta^{\mathsf{ML}}\right)$$

where β^{ML} is the pattern describing the substitution β. Since $u =_{E_\Omega\cup B_\Omega} v_i\beta$, it follows that $\exists\overline{x}:\overline{s}.\, u \subseteq \exists\overline{y_i}:\overline{s_i}.\, v_i$ and therefore only the constraint part must hold.

In the sequel, we abusively use $\textsc{match}(u, \{v_i \mid i \in I\})$ to denote the set of equalities defined by it as well. With this notation we can write $\mathsf{CONS}(F, \Omega, E, B, E_\Omega, B_\Omega) \models \textsc{match}(u, \{v_i \mid i \in I\})$ to express that $\textsc{match}(u, \{v_i \mid i \in I\})$ includes indeed the desired set of matching results.

We can prove now that the two ML patterns for subsumption are equivalent:

Theorem 7.5. *If* $\mathsf{CONS}(F, \Omega, E, B, E_\Omega, B_\Omega) \models \textsc{match}(u, \{v_i \mid i \in I\})$ *then*

$$\mathsf{CONS}(F, \Omega, E, B, E_\Omega, B_\Omega) \vdash$$

$$(\exists \overline{x}:\overline{s}.\, u \wedge \varphi) \subseteq \left(\bigvee_{i\in I} \exists \overline{y_i}:\overline{s_i}.\, v_i \wedge \psi_i\right) \leftrightarrow \left(\varphi \to \bigvee_{(i,\beta)\in\textsc{match}(u,\{v_i \mid i\in I\})} (\psi_i \wedge \beta^{\mathsf{ML}})\right)$$

Proof Sketch. The key property is that of the match result (i, β), which satisfies that $u =_{E_\Omega \cup B_\Omega} \beta(v_i)$. In other words, β is the logical constraint that states that u can be matched by v_i. Thus, the reasoning is as follows. Intuitively, the LHS holds when $u \wedge \varphi$ is $\bot$, i.e., φ is $\bot$, or when u can be matched by v_i for some i. This yields the RHS, which states that if φ holds, then there exists i such that u is matched by the constraint term pattern $v_i|\psi_i$. The matching part is equivalent to the logical constraint β given by the matching function $\textsc{match}$, and ψ_i is the logical constraint in the original constraint term pattern. Both need to be satisfied, and thus we have $\psi_i \wedge \beta^{\mathsf{ML}}$ on the RHS. □

7.1.2. Over-approximating complements

Since matching logic has negation, the difference $[\![u|\top]\!] \setminus [\![u|\varphi]\!]$ is the same with the interpretation in $\mathsf{CONS}F, \Omega, E, B, E_\Omega, B_\Omega$ of the matching logic pattern

$$(\exists \overline{x}:\overline{s}.\, u) \wedge \neg(\exists \overline{x}:\overline{s}.\, (u \wedge \varphi))$$

The predicate $[\![u|\top]\!]$ is the same with the interpretation in $\mathsf{CCP}(F, \Omega, E, B, E_\Omega, B_\Omega)$ of the pattern $\exists \overline{x}:\overline{s}.\, u$, where $\overline{x}:\overline{s}$ is the set of variables occurring in u, and constructor predicate $[\![u|\neg\varphi]\!]$ is the same with the interpretation of $\exists \overline{x}:\overline{s}.\, (u \wedge \neg\varphi)$.

The difference $[\![u|\varphi]\!] \setminus [\![u|\psi]\!]$ is the same as the interpretation of the pattern

$$\exists \overline{x}:\overline{s}.\, (u \wedge \varphi) \wedge \neg(\exists \overline{x}:\overline{s}.\, (u \wedge \psi))$$

and $[\![u|\varphi]\!] \setminus\!\setminus [\![u|\psi]\!]$ is the same as the interpretation of

$$\exists \overline{x}:\overline{s}.\, (u \wedge \varphi) \wedge \exists \overline{x}:\overline{s}.\, (u \wedge \neg\psi),$$

which is equivalent to $\exists \overline{x}:\overline{s}.\, (u \wedge \varphi \wedge \neg\psi)$. We can prove that $[\![u|\varphi]\!] \setminus\!\setminus [\![u|\psi]\!]$ is indeed an over-approximation of the difference:

Theorem 7.6. *The following holds:*

$$\mathsf{CONS}(F, \Omega, E, B, E_\Omega, B_\Omega) \vdash \exists \overline{x}:\overline{s}.\, (u \wedge \varphi) \wedge \neg(\exists \overline{x}:\overline{s}.\, (u \wedge \psi)) \subseteq \exists \overline{x}:\overline{s}.\, (u \wedge \varphi \wedge \neg\psi)$$

Proof Sketch. If φ is $\bot$ then the LHS is $\bot$ and the inclusion obviously hold. If φ is $\top$ and ψ is $\bot$ then $u \wedge \psi$ is $\bot$ and both LHS and RHS are equal to u. If φ is $\top$ and ψ is $\top$ then $\neg(u \wedge \psi)$ is $\neg u$ and hence the LHS is $\bot$ (because $u \wedge \varphi$ is u). □

7.1.3. Parameterized intersections

For the case when $E = B = \emptyset$, it is shown in [17] that

$$\mathsf{TERMALGEBRA}(S, \le, F) \vdash u \wedge v \leftrightarrow u \wedge \alpha^{\mathsf{ML}}$$

where α is the most general unifier of u and v. We obtain as a consequence that $(u \wedge \varphi) \wedge (v \wedge \psi)$ is equivalent to $u \wedge \alpha^{\mathsf{ML}} \wedge \varphi \wedge \psi$, which is the matching logic translation of the corresponding constrained constructor pattern $(u|\varphi) \wedge_Y (v|\psi)$. Moreover, a proof object can be supplied for this equivalence [17]. We claim that this result can be generalized:

Theorem 7.7. *If* $\{\alpha_1, \ldots, \alpha_k\}$ *is a complete set of* $E_\Omega \cup B_\Omega$*-unifiers for* u_1 *and* u_2*, then* $\mathsf{INITIALALGEBRA}(S, \le, \Omega, E_\Omega \cup B_\Omega) \models (u_1 \wedge u_2) \leftrightarrow (u_i \wedge \exists \overline{x}:\overline{s}.\, (\alpha_1^{\mathsf{ML}} \vee \cdots \vee \alpha_k^{\mathsf{ML}}))$*, for* $i = 1, 2$*, where* $\overline{x}:\overline{s}$ *are the new variables in* $\alpha_1, \ldots, \alpha_k$*.*

Proof. Let ρ be an evaluation. By Theorem 6.7 $\overline{\rho}(u_1)$ and $\overline{\rho}(u_2)$ are $(E_\Omega \cup B_\Omega)$-equivalences classes of ground terms (up to isomorphism). If $\overline{\rho}(u_1) = \overline{\rho}(u_2)$ then there is α_i and ρ' such that $\alpha_i(u_1) =_{E_\Omega \cup B_\Omega} \alpha_i(u_2)$ and $\rho = \rho' \circ \alpha_i$. We obviously have $\overline{\rho}(\exists \overline{x}:\overline{s}.\, \alpha_i^{\mathsf{M}})$ equal to $\top$. If $\overline{\rho}(u_1) \ne \overline{\rho}(u_2)$ then $\overline{\rho}(\exists \overline{x}:\overline{s}.\, \alpha_i^{\mathsf{M}})$ is $\bot$, for any $i \in \{1, \ldots, k\}$: otherwise, if $\overline{\rho}(\exists \overline{x}:\overline{s}.\, \alpha_i^{\mathsf{M}})$ is $\top$ for some i, then $\overline{\rho}(u_1 \wedge \exists \overline{x}:\overline{s}.\, \alpha_i^{\mathsf{M}}) = \overline{\rho}(u_2 \wedge \exists \overline{x}:\overline{s}.\, \alpha_i^{\mathsf{M}})$. □

Remark 7.8. The proof of Theorem 7.7 uses the semantics. The proof object for the case $E = B = \emptyset$ is obtained by inspecting the execution trace of the syntactic unification algorithm, which can be seen as a sequence of pattern transformations. This is not always possible for unification modulo $E_\Omega \cup B_\Omega$ and other approaches should be investigated for this case.

So, the parameterized intersection $(u|\varphi) \wedge_Y (v|\psi)$ of two constrained constructor patterns is encoded in matching logic by the conjunction of the corresponding matching logic patterns, provided that $\mathit{Unif}_{E_\Omega \cup B_\Omega}(u, v)$ is known:

$$\bigvee_{\alpha \in \mathit{Unif}_{E_\Omega \cup B_\Omega}(u,v)} \left(u \wedge \varphi \wedge \psi \wedge \alpha^{\mathsf{ML}}\right) \quad \text{(equivalent to } u \wedge \varphi \wedge \psi \wedge \left(\bigvee_{\alpha \in \mathit{Unif}_{E_\Omega \cup B_\Omega}(u,v)} \alpha^{\mathsf{ML}}\right)\text{)}$$

7.1.4. *Parameterized containment*

Let us discuss the matching logic counterpart of the parameterized subsumption. The pattern expressing parameterized containment $[\![u|\varphi]\!] \subseteq_Z \bigcup_{i \in I} [\![(v_i|\psi_i)]\!]$ is

$$\forall \overline{z}:\overline{s'}.\left(\exists \overline{x}:\overline{s}.\, u \wedge \varphi \subseteq \bigvee_{i \in I} \exists \overline{y_i}:\overline{s_i}.\, v_i \wedge \psi_i\right)$$

where $\overline{z}:\overline{s'}$ is the sequence of shared variables Z freely occurring in both $u|\varphi$ and $\{(v_i|\psi_i) \mid i \in I\}$, $\overline{x}:\overline{s}$ is the set of variables different of $\overline{z}:\overline{s'}$ that freely occur in $u|\varphi$, and $\overline{y_i}:\overline{s_i}$ is the set of variables different of $\overline{z}:\overline{s'}$ that freely occur in $v_i|\psi_i$.

The matching logic translation of $u|\varphi \sqsubseteq_Z \{(v_i|\psi_i) \mid i \in I\}$ is

$$\varphi \to \bigvee_{(i,\beta) \in \textsc{match}(u,\{v_i | i \in I\}, Z)} \left(\psi_i \wedge \beta^{\mathsf{ML}}\right)$$

where $\textsc{match}(u, \{v_i \mid i \in I\}, Z)$ is a set of substitutions β defined over $\mathit{var}(v_i) \setminus Z$, and β^{ML} is the pattern describing the substitution β. We can prove now that the two matching logic patterns are equivalent.

Theorem 7.9. *If* $\mathsf{CONS}(F, \Omega, E, B, E_\Omega, B_\Omega) \models \textsc{match}(u, \{v_i \mid i \in I\})$ *then*

$$\mathsf{CONS}(F, \Omega, E, B, E_\Omega, B_\Omega) \vdash \left(\forall \overline{z}:\overline{s'}.\left(\exists \overline{x}:\overline{s}.\, u \wedge \varphi \subseteq \bigvee_{i \in I} \exists \overline{y_i}:\overline{s_i}.\, v_i \wedge \psi_i\right)\right) \leftrightarrow$$
$$\left(\varphi \to \bigvee_{(i,\beta) \in \textsc{match}(u,\{v_i | i \in I\}, Z)} \left(\psi_i \wedge \beta^{\mathsf{ML}}\right)\right)$$

Proof. The main idea is the same as Theorem 7.5 and to use the property (i, β); that is, $u =_{E_\Omega \cup B_\Omega} v_i\beta$ for any shared variables $z_i \in Z$, explaining the quantifier $\forall \overline{z}:\overline{s'}$ that appear on top of the LHS. $\square$

Remark 7.10. One question that arises here is: Why only the static part (given by the constrained constructor patterns) and not the entire reachability logic of rewrite theories? In [8] it is shown that the reachability logic, where the static component (state or configuration) model is given by a matching logic theory (without modal operators) and the transitions are modeled with the special "all-path next" operator $\circ$, can be defined in matching logic. In this paper we showed that constrained constructor patterns can be defined in matching logic. It follows that the entire reachability logic of rewrite theories can be defined in matching logic. Intuitively, the operator $\circ$ for this logic is

$$\circ(v \wedge \varphi) \equiv \exists u. \forall v'.\, \langle u, v\rangle \in \mathit{rew1}_R \to v' \in (v \wedge \varphi),$$

i.e., $\circ(v \wedge \varphi)$ denotes the set of states whose all successors are in $v \wedge \varphi$ (R includes the rewrite rules describing transitions). The partial interpretation of a reachability formula $(u \wedge \varphi) \longrightarrow^{\circledast} (v \wedge \psi)$ is expressed by the matching logic formula $(u \wedge \varphi) \to \nu X.\,(v \wedge \psi) \vee \circ X$.

8. Conclusion

The paper establishes the exact relationship between two approaches that formalize state predicates of distributed systems: constrained constructor patterns [2] and matching logic [9]. The main conclusion from this comparison is that there is a mutual benefit. Matching logic can benefit from borrowing or capturing the computationally efficient algorithms on constrained constructor patterns. Since the matching, unification, and narrowing can be expressed as matching logic patterns, it follows to see how their algorithms can be "instructed" to produce proof objects for the relationship between the transformed patterns. A first step is given in [17], but we think that there is more potential that can be exploited. On the other hand, the theory of constrained constructor patterns can get more expressiveness from its formalization as a fragment of the matching logic. The inference system for reachability logic over rewrite theories given in [2] is simple, but given at a high level and therefore requires a set of operations defined at the semantic level. In this paper we showed that all these operations can be fully axiomatized in matching logic, which open the door towards a complete matching logic axiomatization of the approach given in [2].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The work presented in this paper was supported in part by NSF CNS 16-19275. This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-18-C-0092.

References

[1] S. Skeirik, A. Stefanescu, J. Meseguer, A constructor-based reachability logic for rewrite theories, in: Proceedings of the 27th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2017), in: Lecture Notes in Computer Science, vol. 10855, Springer, Namur, Belgium, 2018, pp. 201–217.
[2] S. Skeirik, A. Stefanescu, J. Meseguer, A constructor-based reachability logic for rewrite theories, Fundam. Inform. 173 (4) (2020) 315–382, https://doi.org/10.3233/FI-2020-1926.
[3] J. Meseguer, Twenty years of rewriting logic, J. Log. Algebraic Program. 81 (7) (2012) 721–781, https://doi.org/10.1016/j.jlap.2012.06.003.
[4] K. Futatsugi, Fostering proof scores in CafeOBJ, in: Proceedings of the 12th International Conference on Formal Engineering Methods (ICFEM 2010), in: Lecture Notes in Computer Science, vol. 6447, Springer, Shanghai, China, 2010, pp. 1–20.
[5] S. Escobar, R. Sasse, J. Meseguer, Folding variant narrowing and optimal variant termination, J. Log. Algebraic Program. 81 (7–8) (2012) 898–928, https://doi.org/10.1016/j.jlap.2012.01.002.
[6] J. Meseguer, Variant-based satisfiability in initial algebras, Sci. Comput. Program. 154 (2018) 3–41, https://doi.org/10.1016/j.scico.2017.09.001.
[7] G. Roşu, Matching logic, Log. Methods Comput. Sci. 13 (4) (2017) 1–61.
[8] X. Chen, G. Rosu, Matching μ-logic, in: Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2019), IEEE, Vancouver, Canada, 2019, pp. 1–13.
[9] X. Chen, G. Roşu, Applicative matching logic, Tech. Rep., http://hdl.handle.net/2142/104616, University of Illinois at Urbana-Champaign, 2019.
[10] X. Chen, Z. Lin, M.-T. Trinh, G. Roşu, Towards a trustworthy semantics-based language framework via proof generation, in: Proceedings of the 33rd International Conference on Computer-Aided Verification, ACM, Los Angeles, USA, 2021, pp. 1–22.
[11] X. Chen, D. Lucanu, G. Rosu, Connecting constrained constructor patterns and matching logic, in: S. Escobar, N. Martí-Oliet (Eds.), Rewriting Logic and Its Applications - 13th International Workshop, WRLA 2020, Virtual Event, October 20-22, 2020, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 12328, Springer, 2020, pp. 19–37.
[12] X. Chen, D. Lucanu, G. Roşu, Initial algebra semantics in matching logic, Tech. Rep., http://hdl.handle.net/2142/107781, University of Illinois at Urbana-Champaign, July 2020.
[13] X. Chen, G. Roşu, Matching mu-logic: foundation of k framework, in: Proceedings of the 8th Conference on Algebra and Coalgebra in Computer Science (CALCO'19), in: Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, vol. 139, 2019, pp. 1–4.
[14] X. Chen, G. Rosu, SETSS'19 lecture notes on K, in: J. Bowen, Z. Liu (Eds.), Engineering Trustworthy Software Systems, in: Lecture Notes in Computer Science, Springer, 2019.
[15] D. Lucanu, V. Rusu, A. Arusoaie, A generic framework for symbolic execution: a coinductive approach, J. Symb. Comput. (2016), https://doi.org/10.1016/j.jsc.2016.07.012.
[16] S. Escobar, J. Meseguer, R. Sasse, Variant narrowing and equational unification, Electron. Notes Theor. Comput. Sci. 238 (3) (2009) 103–119.
[17] A. Arusoaie, D. Lucanu, Unification in matching logic, in: Formal Methods—The Next 30 Years, Porto, Portugal, in: Lecture Notes in Computer Science, vol. 11800, 2019, pp. 502–518.
[18] J.A. Goguen, J. Meseguer, Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations, Theor. Comput. Sci. 105 (2) (1992) 217–273, https://doi.org/10.1016/0304-3975(92)90302-V.
[19] J. Meseguer, Membership algebra as a logical framework for equational specification, in: F. Parisi-Presicce (Ed.), Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97, Tarquinia, Italy, June 1997, Selected Papers, in: Lecture Notes in Computer Science, vol. 1376, Springer, 1997, pp. 18–61.
[20] J. Meseguer, Generalized rewrite theories, coherence completion, and symbolic methods, J. Log. Algebraic Methods Program. 110 (2020), https://doi.org/10.1016/j.jlamp.2019.100483.
[21] X. Chen, G. Rosu, A general approach to define binders using matching logic, Proc. ACM Program. Lang. 4 (ICFP) (2020) 88, https://doi.org/10.1145/3408970.
[22] X. Chen, D. Lucanu, G. Roşu, Matching logic explained, J. Log. Algebraic Methods Program. 120 (2021) 100638, https://doi.org/10.1016/j.jlamp.2021.100638, http://www.sciencedirect.com/science/article/pii/S2352220821000018.
[23] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, Pac. J. Math. 5 (2) (1955) 285–309.
[24] D. Kozen, Results on the propositional μ-calculus, Theor. Comput. Sci. 27 (3) (1983) 333–354, https://doi.org/10.1016/0304-3975(82)90125-6.
[25] A. Kaposi, A. Kovács, T. Altenkirch, Constructing quotient inductive-inductive types, Proc. ACM Program. Lang. 3 (POPL) (Jan. 2019), https://doi.org/10.1145/3290315.
[26] M.P. Fiore, A.M. Pitts, S.C. Steenkamp, Constructing infinitary quotient-inductive types, in: J. Goubault-Larrecq, B. König (Eds.), Proceedings of the 23rd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'20) Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS'20), in: Lecture Notes in Computer Science, vol. 12077, Springer, Dublin, Ireland, 2020, pp. 257–276.
[27] J. Meseguer, Conditioned rewriting logic as a united model of concurrency, Theor. Comput. Sci. 96 (1) (1992) 73–155, https://doi.org/10.1016/0304-3975(92)90182-F.

[28] A. Ştefănescu, Ş. Ciobâcă, R. Mereuţă, B.M. Moore, T.F. Şerbănuţă, G. Roşu, All-path reachability logic, in: RTA-TLCA, in: LNCS, vol. 8560, 2014, pp. 425–440.
[29] G. Roşu, A. Ştefănescu, Ş. Ciobâcă, B.M. Moore, One-path reachability logic, in: LICS 2013, 2013, pp. 358–367.