

## DOMAIN MU-CALCULUS

GUO-QIANG ZHANG<sup>1</sup>

**Abstract.** The basic framework of domain  $\mu$ -calculus was formulated in [39] more than ten years ago. This paper provides an improved formulation of a fragment of the  $\mu$ -calculus without function space or powerdomain constructions, and studies some open problems related to this  $\mu$ -calculus such as decidability and expressive power. A class of language equations is introduced for encoding  $\mu$ -formulas in order to derive results related to decidability and expressive power of non-trivial fragments of the domain  $\mu$ -calculus. The existence and uniqueness of solutions to this class of language equations constitute an important component of this approach. Our formulation is based on the recent work of Leiss [23], who established a sophisticated framework for solving language equations using Boolean automata (a.k.a. alternating automata [12, 35]) and a generalized notion of language derivatives. Additionally, the early notion of even-linear grammars is adopted here to treat another fragment of the domain  $\mu$ -calculus.

**1991 Mathematics Subject Classification.** 03B70, 68Q45, 68Q55.

## INTRODUCTION

### 0.1. DOMAIN LOGICS

Propositional domain logic (a.k.a. Abramsky logic), based on the view of types as topological spaces, properties as open sets, and computational processes as points, provides a smooth integration among three relatively independent approaches to programming semantics: operational, denotational, and axiomatic [1, 39]. In addition to proof systems for higher-order strict-analysis [18] and concurrent processes [3], it has also been adapted to reasoning about imperative parallel programs [10, 39]. The beauty of this approach is that one can pass from the

---

<sup>1</sup> Department of Electrical Engineering and Computer Science  
Case Western Reserve University  
Cleveland, OH 44106, U. S. A.  
gqz@eecs.cwru.edu

denotation of a computational process to its properties, with harmony guaranteed by Stone-style-duality [19]. Moreover, higher-order objects are treated exactly the same way as first-order objects.

The domain  $\mu$ -calculus due to the author [39] is a natural least fixed-point extension of propositional domain logic. This extension is a necessary step to increase the expressive power of propositional domain logic, so that “infinite behavior” such as safety and liveness can be expressed (propositional formulas represent *compact* Scott open sets only, which cannot express such infinite behavior). This  $\mu$ -calculus consists of three syntactic categories: a language of types, a language of formulas, and a proof system with equational rules indexed over the types. In the domain  $\mu$ -calculus, every closed type expression determines a canonical domain, and hence a topological space of Scott open sets. The semantics of a  $\mu$ -formula is a *fixed* Scott open set of the corresponding domain (some standard restrictions are necessary for function space to work properly within Scott open sets), with the open set being the least fixed-point of the operator induced by the given  $\mu$ -formula.

The equational proof system is intended to capture the containment of Scott open sets; it uses Park’s rules [29] for inequality involving least fixed-point formulas (see [13,20,32] as well). The system is said to be *sound* and *complete* if “theorems” of the form  $\varphi \leq \psi$  coincide with their semantic counterparts  $\llbracket \varphi \rrbracket \subseteq \llbracket \psi \rrbracket$ . It is important to note that for each closed type, we have a corresponding  $\mu$ -calculus; therefore, by “domain  $\mu$ -calculus” we refer to *a spectrum of  $\mu$ -calculi* which may or may not share the same properties, such as completeness and decidability.

## 0.2. MODAL $\mu$ -CALCULUS AND DOMAIN $\mu$ -CALCULUS

Many properties of hardware and software systems can be expressed concisely in the propositional modal  $\mu$ -calculus [20], distinct from the domain  $\mu$ -calculus. The modal  $\mu$ -calculus has attracted a great deal of attention in the last decade. Although decidability and finite-model properties have been established early on, the difficult completeness problem was settled only recently [36]. The expressive power of the  $\mu$ -calculus has been studied in [4,9,17,25,26], establishing the strictness of the alternation hierarchy (reference [9] is based on rather sophisticated results of definability [24]).

Domain  $\mu$ -calculus and modal  $\mu$ -calculus share at least two common ideas. One is that they are intended to capture *infinitary* behavior of a system. The other is that fixed-point formulas serve as a uniform way to *approximate* ideal infinitary properties by finite approximations. While the modal  $\mu$ -calculus is based on the Kripke semantics, *domain  $\mu$ -calculus* uses Scott open sets while providing the integration of types, higher-order objects, and denotational semantics and program logics, all in the same framework. These offer a uniform and yet highly flexible set of logical tools, whose application potential is yet to be fully explored.

While much progress has been made for the modal  $\mu$ -calculus, not much is known about the domain  $\mu$ -calculus. The following table provides a brief summary of the situation. It also gives an indication that there is no obvious way to reduce properties of domain  $\mu$ -calculus to those of modal  $\mu$ -calculus due in part to the

lack of finite model property on the domain  $\mu$ -calculus and to the mismatches in the basic setup.

	Modal $\mu$ -calculus	Domain $\mu$ -calculus
“fragment” means	restriction on formulas	restriction on types
finite model property	yes [20]	no
decidability	yes [20]	open
completeness	yes [36]	open
expressiveness	settled [4, 9, 17, 26]	open

In fact, the finite model property of domain  $\mu$ -calculus somewhat depends on the level of abstraction. The answer is “no” because in general, a Scott open set is not compact in the topological sense; the answer would be “yes” from the point of view that every non-empty Scott open set contains a compact element of the underlying domain (if we think of each point of the domain as a “model”).

### 0.3. MAIN CONTRIBUTIONS

The main idea of this work is to establish an interplay between domain logic and automata theory in order to obtain decidability properties of the domain  $\mu$ -calculus. The idea may work in two directions: (i) if a fragment of  $\mu$ -formulas can be encoded as a class of formal languages, and this class of formal languages is decidable, then the domain  $\mu$ -calculus is decidable (for properties such as emptiness and containment); (ii) if, on the other hand, an undecidable class of formal languages (such as context-free languages) can be faithfully embedded (*faithful* in the sense that the translation is a one-to-one function) as  $\mu$ -formulas of a specific type, then the  $\mu$ -calculus for that type (and any type more expressive than that) is undecidable.

The results reported in this paper are of the first kind. We show that (here  $\oplus$  stands for coalesced-sum, and  $\otimes$  for smash-product)

- the domain  $\mu$ -calculus for  $P = \Sigma_{\perp} \oplus \Sigma_{\perp} \otimes P$  is decidable, where  $\Sigma$  is a non-empty, finite set, and  $\Sigma_{\perp}$  the corresponding flat domain. It is equivalent in expressive power to *regular languages* (without the empty string).
- the domain  $\mu$ -calculus for  $Q = \Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes Q \otimes \Sigma_{\perp})$  is decidable. It is equivalent in expressive power to *even linear languages* [5, 6] without containing the empty string.

The rest of the paper is organized as follows. Section 2 provides a formulation of domain  $\mu$ -calculus. Section 3 recalls results on the  $\mu$ -calculus for the domain of natural numbers. Section 4 reviews the notions of Boolean automata and language equations, and identifies a new class of language equations based on the notion of  $\epsilon$ -property, to be used in subsequent sections. Section 5 links a fragment of domain  $\mu$ -calculus to the class of regular languages and proves that this fragment is decidable. Section 6 gives a short summary of the so-called *even linear* languages and shows the decidability of a more general fragment of domain  $\mu$ -calculus. Section

7 provides comments on other possible interplay between automata-theory and domain  $\mu$ -calculus, and points to directions of further development.

## 1. THE DOMAIN $\mu$ -CALCULUS

There are three components of the domain  $\mu$ -calculus: a language of types, a language of formulas, and a collection of proof rules indexed over the types. The domain  $\mu$ -calculus is built on top of the propositional domain logic; the basic step up is therefore analogous to the propositional version [1]. The difference lies in the addition of  $\mu$ -formulas, denoting certain least fixed-points as explained later.

### 1.1. SYNTAX

For the purpose of this paper, we consider the following language of type expressions:

$$\sigma ::= \mathbf{1} \mid \sigma \otimes \tau \mid \sigma \oplus \tau \mid \sigma_{\perp} \mid t \mid \text{rect}.\sigma$$

where  $t$  is a type variable, and  $\sigma, \tau$  range over type expressions. Each *closed type*  $\sigma$  can be interpreted as a Scott domain  $D_{\sigma}$  in the standard way, with  $\mathbf{1}$  as the one-point cpo,  $\otimes$  as smash product,  $\oplus$  as coalesced sum,  $(\ )_{\perp}$  as lifting, and  $\text{rect}.\sigma$  as a recursively defined domain [30] (see next subsection as well).

Other possible type constructors such as function space and powerdomains can also be used, but the relatively small number of type constructors allows us to focus on some fundamental aspects of the  $\mu$ -calculus. We use smash product instead of cartesian product for convenience of relating to formal languages later on.

The  $\mu$ -formulas (henceforth formulas) of a closed type are defined inductively according to the following clauses:

- $t, f, x_0, x_1, \dots$  are formulas of any type  $\sigma$ , with  $t$  for *true*,  $f$  for *false*, and  $\{x_i\}$  a countable collection of variables (of formulas). These formulas will be treated *polymorphically*, so that the same piece of syntax may have different types according to the context in which it appears. To emphasize the type-specific nature of these formulas especially when dealing with the interaction of formulas from different types, we sometimes make the type  $\sigma$  explicit, using superscripts, such as  $t^{\sigma}, x^{\sigma}$ , etc.
- If  $\varphi$  is a formula of type  $\sigma$  and  $\psi$  a formula of type  $\tau$ , then  $\varphi_{\uparrow}$  is a formula of type  $\sigma_{\perp}$ ,  $\varphi \cdot \psi$  is a formula of type  $\sigma \otimes \tau$ , and  $\text{inl} \varphi, \text{inr} \psi$  are formulas of type  $\sigma \oplus \tau$ .
- If  $\varphi$  is a formula of type  $\sigma[\text{rect}.\sigma/t]$ , a type obtained by substituting all the free occurrences of  $t$  in  $\sigma(t)$  by  $\text{rect}.\sigma$ , then  $\varphi$  is a formula of  $\text{rect}.\sigma$ .
- If  $\varphi, \psi$  are formulas of  $\sigma$ , then  $\varphi \wedge \psi, \varphi \vee \psi$ , and  $\mu x^{\sigma}.\varphi$  are formulas of  $\sigma$ .

A formula is called *closed* if it is free of variables. It is called *propositional* if it is free of the least fixed-point operator.

**Notation.** We write  $\mathcal{V}$  for the set of formula variables of all closed types,  $\mathcal{L}_{\sigma}$  for the set of formulas of type  $\sigma$ , and  $\mathcal{L}$  for the set of formulas of all closed types.

**Example.** The cpo  $\mathbf{N}_\perp$  of natural numbers is defined by the type  $\text{rec } t.(\mathbf{1}_\perp \oplus t)$ . Formulas of this type include:

- $\text{inl } t_1$ , which denotes the set  $\{0\}$  and will be abbreviated as  $0$ ,
- $\text{inr } \text{inr } 0$ , which denotes the set  $\{2\}$ , and
- $\mu x.(0 \vee \text{inr } \text{inr } x)$ , which denotes the set of even numbers.

As can be seen from this example, the syntax can get clumsy, and it is necessary to create abbreviations when working with concrete examples. To further alleviate the notational burden, we drop type superscripts/subscripts when contexts permit us to do so.

## 1.2. SEMANTICS

The basic terminology and results of domain theory, especially those related to recursively defined domains, are taken for granted (see [2, 16, 30, 37, 39]). However, since the construction of coalesced sum, and smash product [30] are important for this paper, they are briefly recalled to make the paper self-contained.

Suppose  $D_1$  and  $D_2$  are cpos.

- The coalesced sum of  $D_1$  and  $D_2$  is the cpo  $D_1 \oplus D_2$ , with the bottom element  $\perp_{D_1 \oplus D_2}$ , and tagged elements of the form  $\langle x_i, i \rangle$  such that  $x_i$  belongs to  $(D_i \setminus \{\perp_{D_i}\})$  for  $i = 1, 2$ . Elements with the same tag inherit the order of their components, while elements with distinct tags are incomparable.
- The smash product of  $D_1$  and  $D_2$  is the cpo  $D_1 \otimes D_2$ , consisting of elements from  $(D_1 \setminus \{\perp_{D_1}\}) \times (D_2 \setminus \{\perp_{D_2}\})$ , ordered coordinatewise, together with the bottom element  $(\perp_{D_1}, \perp_{D_2})$ . The effect of this is that the smash product is the same as the standard cartesian product, except that all elements of the form  $(a, \perp_{D_2})$  and  $(\perp_{D_1}, b)$  are identified with the bottom.

We interpret a formula in  $\mathcal{L}_\sigma$  as an open set in the Scott topology  $\Omega(D_\sigma)$  of domain  $D_\sigma$ . For this purpose, we follow the standard practice in denotational semantics and introduce the notion of *environments*. An environment is a mapping

$$\rho : \mathcal{V} \rightarrow \bigcup_{\sigma} \{K \mid K \in \Omega(D_\sigma)\}$$

such that  $\rho(x^\sigma) \in \Omega(D_\sigma)$  for every variable  $x^\sigma$  of  $\sigma$ . So, an environment is nothing but an assignment of Scott open sets in  $\Omega(D_\sigma)$  to variables of the corresponding type. We write  $\mathcal{E}$  for the set of environments.

We define the semantic functions

$$\llbracket \_ \rrbracket^\sigma : \mathcal{L}_\sigma \rightarrow [\mathcal{E} \rightarrow \Omega(D_\sigma)]$$

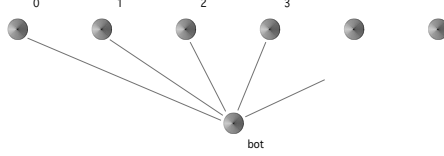
by structural induction, which involves two kinds of clauses: those related to domain constructions (intra-type), and the other related to logical operators (inner-type).

Here are the intra-type clauses, which are more or less standard (see [1, 39]):

- (1)  $\llbracket \varphi \uparrow \rrbracket^{\sigma \perp} \rho$  is the same as  $\llbracket \varphi \rrbracket^{\sigma} \rho$ , except that it now resides in  $D_{\sigma \perp}$  instead of  $D_{\sigma}$ . The new bottom element is not a member of  $\llbracket \varphi \uparrow \rrbracket^{\sigma \perp} \rho$ .
- (2)  $\llbracket \varphi \cdot \psi \rrbracket^{\sigma \otimes \tau} \rho$  is the (set-theoretic) cartesian product  $(\llbracket \varphi \rrbracket^{\sigma} \rho) \times (\llbracket \psi \rrbracket^{\tau} \rho)$ , except that when either  $(a, \perp)$  or  $(\perp, b)$  is a member of  $(\llbracket \varphi \rrbracket^{\sigma} \rho) \times (\llbracket \psi \rrbracket^{\tau} \rho)$ , it is identified with the whole space  $D_{\sigma \otimes \tau}$ .
- (3) When  $\perp_{D_{\sigma}}$  is not a member of  $\llbracket \varphi \rrbracket^{\sigma} \rho$ ,  $\llbracket \text{inl } \varphi \rrbracket^{\sigma \oplus \tau} \rho$  is defined to be the set  $\llbracket \varphi \rrbracket^{\sigma} \rho$  residing in the “left part” of  $D_{\sigma \oplus \tau}$ . Otherwise,  $\llbracket \text{inl } \varphi \rrbracket^{\sigma \oplus \tau} \rho$  is defined to be the whole space  $D_{\sigma \oplus \tau}$ . (The set  $\llbracket \text{inr } \varphi \rrbracket^{\sigma \oplus \tau} \rho$  is defined similarly.)
- (4) Finally, with respect to recursive types,  $\llbracket \varphi \rrbracket^{\text{rec } t, \sigma} \rho$  is defined to be the set

$$\{\epsilon_{\sigma}(u) \mid u \in \llbracket \varphi \rrbracket^{\sigma[(\text{rec } t, \sigma) \setminus t]} \rho\},$$

where  $\epsilon_{\sigma} : D_{\sigma[(\text{rec } t, \sigma) \setminus t]} \rightarrow D_{\text{rec } t, \sigma}$  is the standard isomorphism arising from the solution to the domain equation determined by  $\text{rec } t, \sigma$  (see e.g. [2]).



**Example.** To continue on our earlier example, we provide an illustration of how item 4 above works. Let us briefly discuss why the formula  $\text{inl } t_{\uparrow}$  denotes the set  $\{0\}$ , for the type  $\mathbf{N}_{\perp}$  of natural numbers. Since  $t_{\uparrow}$  is a formula of type  $\mathbf{1}_{\perp}$ ,  $\text{inl } t_{\uparrow}$  is a formula of type  $\mathbf{1}_{\perp} \oplus \mathbf{N}_{\perp}$ . The isomorphism  $\epsilon_{\mathbf{N}_{\perp}} : (\mathbf{1}_{\perp} \oplus \mathbf{N}_{\perp}) \rightarrow \mathbf{N}_{\perp}$  sends  $\perp$  to  $\perp$ , the top of  $\mathbf{1}_{\perp}$  to 0, and  $n$  to the successor of  $n$  in general. Therefore, the interpretation of the formula  $\text{inl } t_{\uparrow}$  is  $\{0\}$ .

With respect to inner-type operators, we have:

- (1)  $\llbracket t \rrbracket^{\sigma} \rho = D_{\sigma}$ ,  $\llbracket f \rrbracket^{\sigma} \rho = \emptyset$ , and  $\llbracket x \rrbracket^{\sigma} \rho = \rho(x)$ .
- (2)  $\wedge$  is interpreted as (set) intersection and  $\vee$  as (set) union.
- (3)  $\llbracket \mu x. \varphi(x) \rrbracket^{\sigma} \rho$  is the least fixed point of the operator  $\Phi$  induced by  $\varphi$  on the complete lattice of Scott open sets over  $D_{\sigma}$ , where

$$\Phi(X) =_{\text{def}} \llbracket \varphi \rrbracket^{\sigma} \rho[x \mapsto X].$$

The last item above needs some justification to ensure that it is well-defined. This relies on the so-called *Fixed-Point Theorem* (sometimes called the Knaster-Tarski Theorem), saying that there exists a least fixed-point for every continuous function  $f$  on a complete partial order, and moreover, the least fixed point is the least upper bound of  $\{f^i(\perp) \mid i \geq 0\}$ .

Now, since our formulas involve neither negation nor any contravariant constructions, the operators they induce on the corresponding lattice of Scott open sets are always continuous. Therefore, the least fixed-point always exists.

**Example.** The most intuitive way to understand the interpretation of a  $\mu$ -formula is through its syntactic unwinding. For example,  $\mu x. (0 \vee \text{inr}^2 x)$  denotes

the set of even numbers because its syntactic unwinding gives:

$$\begin{aligned}
& \mu x.(0 \vee \text{inr}^2 x) \\
& \equiv 0 \vee \text{inr}^2 (\mu x.(0 \vee \text{inr}^2 x)) \\
& \equiv 0 \vee \text{inr}^2 (0 \vee \text{inr}^2 (\mu x.(0 \vee \text{inr}^2 x))) \\
& \equiv 0 \vee (\text{inr}^2 0) \vee \text{inr}^4 (\mu x.(0 \vee \text{inr}^2 x)) \\
& \dots \\
& \equiv 0 \vee (\text{inr}^2 0) \vee (\text{inr}^4 0) \vee \dots \vee (\text{inr}^{2k} 0) \vee \text{inr}^{2k+2} (\mu x.(0 \vee \text{inr}^2 x))
\end{aligned}$$

In general, we can identify the interpretation of  $\mu x.\varphi(x)$  with the infinite union  $\bigcup_{i \geq 0} \llbracket \varphi^i(\mathbf{f}) \rrbracket$ . In other words, we do not need to iterate beyond the first limiting ordinal to reach a fixed-point, as guaranteed by the standard Least Fixed-Point Theorem for continuous functions on domains [37].

### 1.3. EQUATIONAL PROOF SYSTEM

The previous example already suggests the need to recognize the semantic equality between  $\text{inr}(A \vee B)$  and  $(\text{inr}A) \vee (\text{inr}B)$ . To reason about the entailment relation on formulas such as this in general, we describe a proof system consisting of rules (axioms are rules with the empty premise) for reasoning about inequality  $\varphi \leq \psi$  (it is important to note that  $\leq$  is not an operator on formulas). The system is composed of three interacting components: the meta-predicate, the *inner-type* rules, and the *intra-type* rules. Note that one can go back and forth freely (i.e., fixing one completely determines the other) from  $\leq$  to  $=$ , since  $\varphi \leq \psi$  if and only if  $\varphi \wedge \psi = \varphi$ , and  $\varphi = \psi$  if and only if  $\varphi \leq \psi$  and  $\psi \leq \varphi$ .

- There are two meta-predicates for *closed* formulas:  $\mathsf{T}$  for “termination” and  $\mathsf{P}$  for “prime open”. The intuition is that if  $\mathsf{T}(\varphi)$  (and in this case we call  $\varphi$  a *terminating formula*), then  $\perp \notin \llbracket \varphi \rrbracket$ ; if  $\mathsf{P}(\varphi)$ , then  $\llbracket \varphi \rrbracket$  represents a complete prime (non-empty, in particular) in the lattice of Scott open sets  $\Omega(D_\sigma)$  (an open set is a complete prime precisely when it is the up-closure of a single compact element). The two meta-predicates are defined syntactically as follows:
  - $\mathsf{T}(\varphi)$  if every sub-formula  $t$  of  $\varphi$  occurs inside a lifting-context  $(\ )_\uparrow$ .
  - $\mathsf{P}(\varphi)$  if every sub-formula of  $\varphi$  is free of  $\mathbf{f}$ , disjunction, conjunction, and the least fixed-point operator  $\mu$ .

Note that as a derived property, we have  $\frac{\mathsf{T}(\varphi(\mathbf{f}))}{\mathsf{T}(\mu x.\varphi(x))}$  because if every sub-formula  $t$  of  $\varphi(\mathbf{f})$  occurs inside a lifting-context, then every sub-formula  $t$  of  $\mu x.\varphi(x)$  also occurs inside a lifting-context.

- The inner-type proof rules include the standard Boolean axioms for distributivity, commutativity, and associativity. Of particular importance are Park’s rules for reasoning about least fixed-point formulas:

$$\varphi(\mu x.\varphi(x)) \leq \mu x.\varphi(x) \qquad \frac{\psi(\varphi) \leq \varphi}{\mu x.\psi(x) \leq \varphi}$$

- There is a set of intra-type proof rules for each domain construction, which we describe one by one.
  - *Lifting*.  $(f^\sigma)_\uparrow = f^{\sigma^\perp}$ , and  $(\ )_\uparrow$  distributes over  $\wedge$  and  $\vee$ .
  - *Smash product*.
 
$$- f \cdot \psi = \varphi \cdot f = f \quad \frac{P(\psi)}{t \cdot \psi = t} \quad \frac{P(\varphi)}{\varphi \cdot t = t}$$
 -  $\cdot$  distributes over  $\wedge, \vee$  on both left and right.
  - *Coalesced sum*.
 
$$- \text{inl } t = \text{inr } t = t, \quad \text{inl } f = \text{inr } f = f \quad \frac{T(\varphi) \quad T(\psi)}{\text{inl } \varphi \wedge \text{inr } \psi = f}$$
 -  $\text{inl}$  and  $\text{inr}$  distribute over  $\wedge$  and  $\vee$ .
- The foregoing proof rules are quite standard. However, we introduce a set of new rules for *contracting context*, which is indispensable when it comes to reasoning about formulas over a recursive type:

$$\frac{T(p) \quad p \leq \varphi(p)}{p = f} \quad \text{where } \varphi(\bullet) \text{ is a contracting context}$$

Since the notion of contracting context is introduced here the first time for domain logic, it would be useful to formulate it as a definition. The intuition is that a contracting-context is similar to the operator  $\bullet \mapsto a\bullet$ , with  $0 < a < 1$ . For a non-negative real number  $r$ , then,  $r \leq ar$  implies  $r = 0$ .

**Definition 1.1.** According to standard formulation, a context is a formula with one “place-holder” denoted as  $\bullet$  in it. A contracting context is defined inductively as follows.

- (1)  $(\bullet)_\uparrow, \text{inl}(\bullet), \text{inr}(\bullet)$ , are contracting contexts.
- (2)  $p \cdot (\bullet)$  is a contracting context if  $T(p)$ , and similarly,  $(\bullet) \cdot q$  is a contracting context if  $T(q)$ .
- (3) if  $\varphi(\bullet)$  and  $\psi(\bullet)$  are contracting contexts, then so is  $\varphi(\psi(\bullet))$ .

The following are some typical *instances* of rules for contracting contexts (note that the first rule remains valid without the condition  $T(p)$  for lifting):

$$\frac{T(p) \quad p \leq p_\uparrow}{p = f} \quad \frac{T(p) \quad p \leq \text{inr } p}{p = f} \quad \frac{T(p) \quad p \leq \text{inl } p}{p = f}$$

$$\frac{T(p) \quad T(q) \quad p \leq q \cdot p}{p = f} \quad \frac{T(p) \quad T(q) \quad p \leq p \cdot q}{p = f}$$

We have two additional meta-rules which are part of the proof system by default: one is the preservation of  $=$  under “substituting equals by equals”, and the other is the monotonicity of the constructors with respect to  $\leq$ . Hence,  $(\ )_\uparrow, \text{inl}, \text{inr}$  are monotonic operators of arity one, and  $\cdot$  is a monotonic operator of arity two.

A basic result about the *propositional* (i.e., without variables and  $\mu$ -formulas) part of the system is the following.

**Theorem 1.2** (Abramsky [1]). *Propositional domain logic is sound, complete, and decidable.*

The key idea of the proof is to prove that every propositional formula is provably equivalent to a finite disjunction of formulas representing complete primes in the lattice of Scott open sets. Note that the formulation here uses smash product, which requires some care to make sure that, for example, formulas such as  $\varphi \cdot t$  be shown to be equivalent to  $t$  when  $\varphi$  is not equivalent to  $f$ , and to  $f$  when  $\varphi = f$ .

**Proposition 1.3.** *The domain  $\mu$ -calculus is sound.*

*Proof.* The proof can be carried out by examining each rule. We only check the termination predicate related to  $\mu$ -formulas, introduced in this paper for the first time; the rest are routine. It is straightforward to show that for each propositional formula  $p$ ,  $\top(p)$  implies  $\perp \notin \llbracket p \rrbracket$ . We need to show that if every sub-formula  $t$  of  $\mu x.\varphi(x)$  occurs inside a lifting-context, then  $\perp \notin \llbracket \mu x.\varphi(x) \rrbracket$ . First note that if every sub-formula  $t$  of  $\mu x.\varphi(x)$  occurs inside a lifting-context, then every sub-formula  $t$  of  $\varphi^i(f)$  also occurs inside a lifting-context, for each  $i \geq 0$ . Suppose  $\perp \in \llbracket \mu x.\varphi(x) \rrbracket$ . Since  $\llbracket \mu x.\varphi(x) \rrbracket = \bigcup_{i \geq 0} \llbracket \varphi^i(f) \rrbracket$ , we know that  $\perp \in \llbracket \varphi^i(f) \rrbracket$  for some  $i \geq 1$ . This is a contradiction to the propositional case.  $\square$

**Remark.** It is important to note what is not expected to hold in general as well. For example, the associative law for  $\cdot$  does not hold: we do not in general have  $\varphi_1 \cdot (\varphi_2 \cdot \varphi_3) = (\varphi_1 \cdot \varphi_2) \cdot \varphi_3$ . Neither do we have the commutativity between  $\text{inl}$  and  $\text{inr}$ . With respect to  $\mu$ -formulas, we can prove neither  $\mu x.p \wedge \varphi(x) = p \wedge (\mu x.\varphi(x))$ , nor  $\mu x.p \vee \varphi(x) = p \vee (\mu x.\varphi(x))$  in general.

However, while concatenation is not distributive over intersection with respect to languages, we do have  $\varphi \cdot (\psi_1 \wedge \psi_2) = (\varphi \cdot \psi_1) \wedge (\varphi \cdot \psi_2)$ , thanks to the type discipline.

#### 1.4. SOME META-THEOREMS

The fixed-point proof rules together with monotonicity allow us to prove the equality

$$\varphi(\mu x.\varphi(x)) = \mu x.\varphi(x).$$

Several other meta-theorems are also provable (for more examples, see [39]). These are built on the observation that all formulas  $\varphi$  free of the  $\mu$ -operator are *distributive* over both  $\wedge$  and  $\vee$ , i.e.,

- $\varphi(x \vee y) = \varphi(x) \vee \varphi(y)$
- $\varphi(x \wedge y) = \varphi(x) \wedge \varphi(y)$

are deducible from the equational proof system presented in Section 1.3.

**Theorem 1.4.** *In the following,  $p_i$ s are assumed to be closed formulas. We have*

- (1)  $\mu x.(p_1 \vee p_2 \vee \varphi(x)) = [\mu x.p_1 \vee \varphi(x)] \vee [\mu y.p_2 \vee \varphi(y)]$
- (2)  $\mu x. [(\bigvee_{i=0}^n \varphi^i(p)) \vee \varphi^{n+1}(x)] = \mu x.p \vee \varphi(x)$
- (3)  $\varphi(\mu x.p \vee \varphi^n(x)) = \mu x.\varphi(p) \vee \varphi^n(x)$
- (4)  $\mu x.p \vee \varphi^m(x) \vee \varphi^n(x)$   
 $= \bigvee_{i=0}^{m-1} [\mu x.\varphi^{i+n}(p) \vee \varphi^m(x)] \vee \bigvee_{j=0}^{n-1} [\mu y.\varphi^{j+m}(p) \vee \varphi^n(y)]$

$$(5) \quad \mu x. (\varphi^m(x) \vee \mu y. p \vee \varphi^n(y)) = \mu x. p \vee \varphi^m(x) \vee \varphi^n(x)$$

To see how item 1 is derivable, let  $p$  denote  $(\mu x. p_1 \vee \varphi(x)) \vee (\mu y. p_2 \vee \varphi(y))$  and  $q(x)$  denote  $p_1 \vee p_2 \vee \varphi(x)$ . Then by the distributivity of  $\varphi$ , we have (here  $\equiv$  stands for syntactic abbreviation)

$$\begin{aligned} q(p) &\equiv p_1 \vee p_2 \vee \varphi(p) \\ &= p_1 \vee p_2 \vee \varphi((\mu x. p_1 \vee \varphi(x)) \vee (\mu y. p_2 \vee \varphi(y))) \\ &= p_1 \vee p_2 \vee \varphi(\mu x. p_1 \vee \varphi(x)) \vee \varphi(\mu y. p_2 \vee \varphi(y)) \\ &= (p_1 \vee \varphi(\mu x. p_1 \vee \varphi(x))) \vee (p_2 \vee \varphi(\mu y. p_2 \vee \varphi(y))) \\ &= (\mu x. p_1 \vee \varphi(x)) \vee (\mu y. p_2 \vee \varphi(y)) \\ &\equiv p \end{aligned}$$

By Park's rules, we have  $\mu x. q(x) \leq p$ , which provides the non-trivial direction for item 1. Note that the distributivity of  $\varphi$  is used in the second step.

## 2. DOMAIN $\mu$ -CALCULUS FOR $\mathbf{N}_\perp$

In [39] we proved the soundness, completeness, and established the expressive power of the domain  $\mu$ -calculus for  $\mathbf{N}_\perp$  with some mild syntactic restrictions. This brief section serves two purposes. One is that it provides an example of how the new notions of *termination predicate*  $\top$  and *contracting contexts* are used in proving some non-trivial equalities (in [39], two ad hoc proof rules were used to achieve the results without this apparatus). The second is that it gives a brief account of the decidability of the  $\mu$ -calculus for  $\mathbf{N}_\perp$ , which offers intuition on similar approaches used in the subsequent sections based on results from automata theory.

We begin with some examples to show how the equational proof system – the termination predicate and the contracting contexts in particular – is put to use. For conciseness, let's abbreviate  $\text{inl}\top$  as  $0$  and  $\text{inr}$  as  $s$ . By Theorem 1.4, we can derive

$$s(\mu x. (0 \vee s^2(x))) = \mu x. (s(0) \vee s^2(x)),$$

which means that if we apply the successor to the whole set of even numbers we get the set of odd numbers, as expected.

Many other facts about natural numbers are derivable from the  $\mu$ -calculus as special cases of the theorems given in the previous section. Listed below are a few of them:

$$\begin{aligned} \mu x. s(x) &= f, \\ \mu x. 0 \vee s^2(x) &= 0 \vee s^2(\mu x. 0 \vee s^2(x)), \\ \mu x. (0 \vee s(0) \vee s^2(x)) &= (\mu x. 0 \vee s^2(x)) \vee (\mu x. s(0) \vee s^2(x)), \\ \mu x. (0 \vee s(0) \vee s^2(x)) &= \mu x. 0 \vee s(x). \end{aligned}$$

Consider the formula  $(\mu x. 0 \vee s^2(x)) \wedge (\mu x. s(0) \vee s^2(x))$ . It expresses the intersection of even numbers and odd numbers, and hence should give us the empty set, that is, we should be able to prove

$$(\mu x. 0 \vee s^2(x)) \wedge (\mu x. s(0) \vee s^2(x)) = f.$$

The rules associated with the termination predicate and contracting contexts make it a relatively simple task. Note that we have

$$\begin{aligned}
& (\mu x.0 \vee \mathbf{s}^2 x) \wedge (\mu x.\mathbf{s}(0) \vee \mathbf{s}^2 x) \\
&= (\mu x.0 \vee \mathbf{s}^2 x) \wedge \mathbf{s}(\mu x.0 \vee \mathbf{s}^2 x) \\
&= (0 \vee \mathbf{s}^2(\mu x.0 \vee \mathbf{s}^2 x)) \wedge \mathbf{s}(\mu x.0 \vee \mathbf{s}^2 x) \\
&= (0 \wedge \mathbf{s}(\mu x.0 \vee \mathbf{s}^2 x)) \vee (\mathbf{s}^2(\mu x.0 \vee \mathbf{s}^2 x) \wedge \mathbf{s}(\mu x.0 \vee \mathbf{s}^2 x)) \\
&= \mathbf{s}^2(\mu x.0 \vee \mathbf{s}^2 x) \wedge \mathbf{s}(\mu x.0 \vee \mathbf{s}^2 x) \\
&= \mathbf{s}[(\mu x.0 \vee \mathbf{s}^2 x) \wedge (\mu x.\mathbf{s}(0) \vee \mathbf{s}^2 x)]
\end{aligned}$$

Here we used the rule involving the termination predicate,  $\frac{\mathsf{T}(\varphi) \quad \mathsf{T}(\psi)}{\mathsf{inl} \varphi \wedge \mathsf{inr} \psi = \mathbf{f}}$ , to derive  $(0 \wedge \mathbf{s}(\mu x.0 \vee \mathbf{s}^2 x)) = \mathbf{f}$  in the fourth step. This is because we have  $\mathsf{inl} \mathbf{t}_\uparrow \equiv 0$  and  $\mathsf{inr} \equiv \mathbf{s}$ , and  $\mathsf{T}(\mathbf{t}_\uparrow)$ ,  $\mathsf{T}(\mu x.0 \vee \mathbf{s}^2 x)$ , by the definition of the termination predicate. Now since  $\mathbf{s}$  is a contracting context and  $p \equiv (\mu x.0 \vee \mathbf{s}^2 x) \wedge (\mu x.\mathbf{s}(0) \vee \mathbf{s}^2 x)$  is a termination formula (i.e.  $\mathsf{T}(p)$ ), we have, by rule  $\frac{\mathsf{T}(p) \quad p \leq \mathsf{inr} p}{p = \mathbf{f}}$  for contracting-context,

$$(\mu x.0 \vee \mathbf{s}^2 x) \wedge (\mu x.\mathbf{s}(0) \vee \mathbf{s}^2 x) = \mathbf{f}.$$

The decidability of the domain  $\mu$ -calculus for  $\mathbf{N}_\perp$  follows from an effective procedure to determine the semi-linear set represented by each closed formula for  $\mathbf{N}_\perp$ .

**Definition 2.1.** A set of natural numbers is called a semi-linear set if it is of the form  $N_0 \cup \bigcup_{k \in \omega} (kn_0 + N_1)$ , where  $n_0$  is a natural number,  $N_0$  and  $N_1$  are finite sets of natural numbers, and  $k + A \stackrel{\text{def}}{=} \{k + n \mid n \in A\}$  for a natural number  $k$  and a set  $A$ .

Each closed, terminating  $\mu$ -formula  $\varphi$  of  $\mathbf{N}_\perp$  determines a subset of natural numbers  $\mathcal{N}(\varphi)$  by the following inductive definition:

- $\mathcal{N}(\mathbf{f}) = \emptyset$ .
- $\mathcal{N}(\mathsf{inl} \mathbf{t}_\uparrow) = \{0\}$ .
- $\wedge$  corresponds to intersection and  $\vee$  corresponds to union.
- if  $\mathcal{N}(\varphi) = A$ , then  $\mathcal{N}(\mathsf{inr} \varphi) = \{k + 1 \mid k \in A\}$ .
- $\mathcal{N}(\mu x.\varphi(x)) = \bigcup_{i \geq 0} \mathcal{N}(\varphi^i(\mathbf{f}))$ .

Thus  $\mathcal{N}$  is defined in a similar way as  $\llbracket \cdot \rrbracket$ , except that it is only defined for *terminating formulas*  $\varphi$  (i.e. those for which  $\mathsf{T}(\varphi)$  holds).

**Theorem 2.2.** (1) *For each terminating formula  $\varphi$ ,  $\mathcal{N}(\varphi)$  is a semi-linear set, and for every semi-linear set  $A$ , there exists a terminating formula  $\varphi$  such that  $\mathcal{N}(\varphi) = A$ .*  
(2) *There is an effective procedure to find the semi-linear set determined by a terminating formula.*  
(3) *For terminating formulas  $\varphi, \psi$ ,  $\llbracket \varphi \rrbracket \subseteq \llbracket \psi \rrbracket$  if and only if  $\mathcal{N}(\varphi) \subseteq \mathcal{N}(\psi)$ , and the problems of semantic containment  $\llbracket \varphi \rrbracket \subseteq \llbracket \psi \rrbracket$  and emptiness  $\llbracket \varphi \rrbracket = \emptyset$  are decidable.*

Proofs for (1) and (2) can be found in [39], using the notion called *linear shift*. For (3), first note that  $\mathsf{T}$  is an effective predicate for  $\mathbf{N}_\perp$ : there is an algorithm which decides, for each  $\varphi$ , whether  $\mathsf{T}(\varphi)$  holds or not. The procedure for deciding  $\llbracket\varphi\rrbracket \subseteq \llbracket\psi\rrbracket$  consists of the following steps:

- (1) If  $\mathsf{T}(\psi)$  is false, then  $\llbracket\varphi\rrbracket \subseteq \llbracket\psi\rrbracket$  is true.
- (2) If  $\mathsf{T}(\psi)$  but not  $\mathsf{T}(\varphi)$ , then  $\llbracket\varphi\rrbracket \subseteq \llbracket\psi\rrbracket$  is false.
- (3) If both  $\mathsf{T}(\varphi)$  and  $\mathsf{T}(\psi)$ , then  $\llbracket\varphi\rrbracket \subseteq \llbracket\psi\rrbracket$  if and only if  $\mathcal{N}(\varphi) \subseteq \mathcal{N}(\psi)$ , and according to [33] (and in fact, by automata theory), this is decidable.

The emptiness problem is a special case of the general containment problem  $\llbracket\varphi\rrbracket \subseteq \llbracket\psi\rrbracket$ , and hence it is decidable as well.

### 3. BOOLEAN AUTOMATA AND LANGUAGE EQUATIONS

The key idea for the study of expressive power and decidability results of certain fragments of the domain  $\mu$ -calculus is to translate certain types of  $\mu$ -formulas to systems of language equations (for an overview of language equations, see [23]). This section provides a review of relevant results on the interplay between alternating finite automata and language equations that will be used in subsequent sections. The concept of Boolean automata is only implicitly needed for this section; we review this concept nonetheless since it is basic and it provides intuition about the decidability on containments between solutions to a certain class of language equations.

*Alternating finite automata* (AFA, a.k.a. Boolean automata in [11]), are a restricted class of alternating Turing machines as introduced in [12] around roughly the same time as [11]. These finite state machines serve as a useful tool for solving language equations. Our main reference for AFA are [11, 14, 15, 22, 23, 35, 38], and *we will treat Boolean automata and AFA as synonymous terms in this paper*.

Intuitively, an AFA is a finite state machine, whose transition function is a Boolean expression over the states (i.e. a proposition using state labels as atomic variables). The designation of final states amounts to fixing a truth assignment for the states. A string is accepted by an AFA when the “rewriting” of a string from the starting state gives a Boolean expression which evaluates to true under the fixed truth assignment (determined by “final states”). This is captured formally in the next definition.

**Definition 3.1.** A Boolean automaton is a tuple

$$A = (Q, \Sigma, s, \delta, F),$$

where

- $Q$  is a finite set of states,
- $\Sigma$  is the alphabet,
- $s \in Q$  the starting state,
- $\delta$  is a function from  $Q \times \Sigma$  to  $\mathcal{B}(Q)$ , the set of *Boolean expressions* over  $Q$ ,
- $F$  is a function from  $Q$  to  $\{0, 1\}$ .

First, we can extend  $\delta$  to a function from  $\mathcal{B}(Q) \times \Sigma$  to  $\mathcal{B}(Q)$ , where  $\delta(\varphi, a)$  is obtained by simultaneously replacing every state (variable)  $p$  in  $\varphi$  by  $\delta(p, a)$ . Next, the transition function  $\delta$  can be further extended to one from  $\mathcal{B}(Q) \times \Sigma^*$  to  $\mathcal{B}(Q)$ , by induction on strings:

- $\widehat{\delta}(\varphi, \epsilon) := \varphi$  for each  $\varphi \in \mathcal{B}(Q)$ ,
- $\widehat{\delta}(\varphi, wa) := \delta(\widehat{\delta}(\varphi, w), a)$  for  $a \in \Sigma$  and  $w \in \Sigma^*$ .

A string  $w$  is accepted by an AFA  $(Q, \Sigma, s, \delta, F)$  if the Boolean expression  $\widehat{\delta}(s, w)$  evaluates to true under  $F$ . The language determined by an AFA is the set of all strings accepted by it.

The following is a basic fact about AFA (see, e.g., [23, 38]).

**Theorem 3.2.** *There is an effective procedure to construct an equivalent DFA (deterministic finite automaton) for a given AFA. As an immediate consequence of this reduction procedure, the the class of languages accepted by AFA are regular, and the problems of containment and emptiness of languages accepted by AFA are decidable.*

The reduction of an AFA to a DFA and then to a canonical, minimized DFA are the critical steps for deciding the containment of languages accepted by AFA. The first step follows from the same idea of the well-known “powerset” construction [34] to convert an NFA to a DFA. Instead of using the powerset of  $Q$  as the state set, we now use the set of all Boolean *functions*  $f : [Q \rightarrow \{0, 1\}] \rightarrow \{0, 1\}$  as states, of which there are  $2^{2^{|Q|}}$ . Note that each such function  $f$  determines an equivalence class of Boolean expressions. Although the set of Boolean expressions over  $Q$  is infinite, the number of its equivalence classes is finite, i.e.,  $2^{2^{|Q|}}$ . The second step follows from well-known minimization algorithms for DFA.

To be precise, the definition of AFA uses *positive Boolean expressions* (which are syntactic objects) instead of using *Boolean functions* (which are semantic objects). Positive Boolean expressions are those which are free of negation (or complement).

Boolean automata serve as a technical tool for solving language equations [23]. We will be concerned with a class of language equations without complement. But we introduce a slightly more accommodating property, called the  $\epsilon$ -property, to identify a larger class of equation systems with unique solutions.

**Definition 3.3.** A system of language equations is a collection of equations

$$\begin{cases} X_1 = \varphi_1(X_1, \dots, X_n) \\ \dots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{cases}$$

where each  $\varphi_i$  is an expression built up inductively from the following entities:

- variables  $X_i$ ,  $i = 0, \dots, n$ , and constants which stands for regular languages.
- union and intersection.
- left-concatenation [23], i.e., when forming a concatenation, the left operand must be a constant (language).

A language vector  $(L_1, \dots, L_n)$  is said to be a solution of the system of language equations if by substituting the variables  $X_i$  by their corresponding languages  $L_i$  for  $i = 0, \dots, n$ , we obtain language equalities  $L_i = \varphi_i(L_1, \dots, L_n)$  for each  $i = 0, \dots, n$ .

A departure from [23] is that we do not use complement here. But we make intersection a primitive operation, rather than one which can be encoded by union and complement through the standard De Morgan's laws. Another departure from [23] is that we deal with equations of a slightly more general form (without complement), which do not necessarily have the so-called  $\lambda$ -property [23].

We introduce the notion of  $\epsilon$ -property, associated with systems of language equations, rather than individual expressions as in [23].

**Definition 3.4.** With respect to a system of language equations

$$\begin{cases} X_1 = \varphi_1(X_1, \dots, X_n) \\ \dots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{cases}$$

the set of expressions having the  $\epsilon$ -property is defined inductively as follows.

- (1) Any constant has the  $\epsilon$ -property.
- (2) A variable  $X_i$  has the  $\epsilon$ -property if  $\varphi_i(X_1, \dots, X_n)$ , the right-hand-side of the equation  $X_i = \varphi_i(X_1, \dots, X_n)$  from the equation system, has the  $\epsilon$ -property.
- (3) A left-concatenation  $L \cdot \psi$  has the  $\epsilon$ -property if either  $\epsilon \notin L$ , or  $\psi$  has the  $\epsilon$ -property.
- (4) A conjunction has the  $\epsilon$ -property if each of its conjuncts has the  $\epsilon$ -property. Similarly, a disjunction has the  $\epsilon$ -property if each of its disjuncts has the  $\epsilon$ -property.

Finally, the system of language equations above is said to have the  $\epsilon$ -property if every variable  $X_i$  has the  $\epsilon$ -property for  $i = 0, \dots, n$  with respect to this equation system.

Item 2 above is the reason why our notion of  $\epsilon$ -property is equation-system dependent, rather than only variable-dependent, as formulated in [23]. The next example illustrates the difference.

**Example.** Consider the system of language equations

$$\begin{cases} X_1 = L \\ X_2 = X_1 \end{cases}$$

where  $L$  is a constant language. It clearly has a unique solution. It is also clear that according to Definition 3.4, it has the  $\epsilon$ -property. But it fails to have the  $\lambda$ -property because  $X_1$  (the second occurrence) does not have the  $\lambda$ -property with respect to  $X_1$  (see [23], page 42, for the definition of the  $\lambda$ -property).

Although the formulation of  $\epsilon$ -property differs from the  $\lambda$ -property, we still make essential use of [23] (Theorem 4.10, page 52) by reducing a system of equation

with the  $\epsilon$ -property to an equivalent system (in the sense of having the same set of solutions) of equations with the  $\lambda$ -property in order to establish the existence and the uniqueness of solutions.

**Lemma 3.5** (Substitution Lemma). *Consider two equation systems*

$$\begin{cases} X_1 = \varphi_1(X_1, \dots, X_n) \\ \dots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{cases}$$

and

$$\begin{cases} X_1 = \varphi_1(X_1, \dots, \varphi_n(X_1, \dots, X_n)) \\ \dots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{cases}$$

where the second is obtained from the first by substituting an occurrence of  $X_n$  in the first equation by  $\varphi_n(X_1, \dots, X_n)$ . The two systems have the same set of solutions (if any).

The proof follows directly from equational reasoning, by noting that if  $L_n = \varphi_n(L_1, \dots, L_n)$ , then replacing  $L_n$  by  $\varphi_n(L_1, \dots, L_n)$  or vice-versa maintains language equality in any context; the syntactic difference disappears in a solution.

As an immediate consequence of this lemma, we can substitute any variables in any of its occurrences any number of times this way without affecting the solutions.

**Lemma 3.6.** *Suppose*

$$\begin{cases} X_1 = \varphi_1(X_1, \dots, X_n) \\ \dots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{cases}$$

is a system of language equations with the  $\epsilon$ -property. Then there exists a system of language equations

$$\begin{cases} X_1 = \varphi'_1(X_1, \dots, X_n) \\ \dots \\ X_n = \varphi'_n(X_1, \dots, X_n) \end{cases}$$

with the  $\lambda$ -property such that a language vector is a solution for the former if and only if it is a solution for the latter.

*Proof.* We provide an effective procedure to derive a new equation system with the  $\lambda$ -property from a given one with the  $\epsilon$ -property using Lemma 3.5. Suppose

$$\begin{cases} X_1 = \varphi_1(X_1, \dots, X_n) \\ \dots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{cases}$$

is a system of language equations with the  $\epsilon$ -property. Suppose the expression  $\varphi_j(X_1, \dots, X_n)$  does not have the  $\lambda$ -property with respect to  $X_i$ . Then by induction on the structure of  $\varphi_j(X_1, \dots, X_n)$ , there exists an occurrence of the variable  $X_i$  in  $\varphi_j(X_1, \dots, X_n)$  which does not have the  $\lambda$ -property (with respect

to  $X_i$ ), although it has the  $\epsilon$ -property because  $\varphi_i(X_1, \dots, X_n)$  has the  $\epsilon$ -property. (Note that the distinction between the  $\lambda$ -property and the  $\epsilon$ -property stems only from the way variables are treated.) Now substitute this particular occurrence of  $X_i$  by  $\varphi_i(X_1, \dots, X_n)$  in  $\varphi_j(X_1, \dots, X_n)$ , and call the resulting expression  $\varphi'_j(X_1, \dots, X_n)$ . Replace the original equation

$$X_j = \varphi_j(X_1, \dots, X_n)$$

by the equation

$$X_j = \varphi'_j(X_1, \dots, X_n)$$

and keep the rest of the equations the same to form a new system. This reduces the number of occurrences of  $X_i$  in  $\varphi_j(X_1, \dots, X_n)$  with the  $\epsilon$ -property but without the  $\lambda$ -property by 1<sup>1</sup>. Repeat this procedure until all subexpressions on the right-hand-side of the equations have the  $\lambda$ -property with respect to all  $X_i$ s. By Lemma 3.5, the resulting equation system has the same set of solutions as the original one.

Since the  $\epsilon$ -property is defined inductively from the constants and expressions of the form  $L \cdot X$  with  $L$  not containing the empty string, the procedure prescribed above terminates with no expressions on the right-hand-side failing the  $\lambda$ -property.  $\square$

To continue on the earlier example

$$\begin{cases} X_1 = L \\ X_2 = X_1 \end{cases}$$

we obtain, by the procedure described in Lemma 3.6, the following equation system

$$\begin{cases} X_1 = L \\ X_2 = L \end{cases}$$

which clearly has the  $\lambda$ -property because expressions on the right-hand-side of the equations are all constants.

As an immediate consequence of Lemma 3.6 and Theorem 4.10 of [23], we have

**Theorem 3.7.** *If a system of language equations*

$$\begin{cases} X_1 = \varphi_1(X_1, \dots, X_n) \\ \dots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{cases}$$

*has the  $\epsilon$ -property, then it has a unique solution  $(L_1, \dots, L_n)$ , where each  $L_i$  is a regular language for  $i = 0, \dots, n$ . Moreover, there is an effective procedure for the*

---

<sup>1</sup>Note that by Definition 3.4, the reason that  $X_i$  has the  $\epsilon$ -property is due to the fact that  $\varphi_i(X_1, \dots, X_n)$  has the  $\epsilon$ -property in lack of knowing  $X_i$ 's  $\epsilon$ -property, by the inductive nature of Definition 3.4. In other words, the only way to establish the  $\epsilon$ -property of  $X_i$  is to establish the  $\epsilon$ -property of  $\varphi_i(X_1, \dots, X_n)$  first, without using the information that  $X_i$  has the  $\epsilon$ -property.

construction of a Boolean automaton such that  $L_i$  is the language accepted by the automaton by selecting a distinct starting state for each  $i = 0, \dots, n$ .

#### 4. $\mu$ -CALCULUS FOR $P = \Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes P)$

This section establishes a correspondence between ( $\epsilon$ -free, or, not containing the empty string) regular languages over  $\Sigma$  (the alphabet set), and closed  $\mu$ -formulas of type  $\text{rec } t.\Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes t)$ . This correspondence allows us to characterize the expressive power and show decidability results for the domain  $\mu$ -calculus.

To begin with, a set  $\Sigma$  of size  $n$  can be represented as the coalesced sum  $\mathbf{1}_{\perp} \oplus (\mathbf{1}_{\perp} \oplus (\mathbf{1}_{\perp} \oplus \dots))$  with  $(n - 1)$  times of  $\oplus$  operations. Each distinct symbol of  $\Sigma$  can then be uniquely identified with a formula  $\text{inr}^k \text{inl } t_{\uparrow}$  for some  $k \geq 0$ . For convenience, we use standard symbols such as  $a, b, c$  to range over both elements of  $\Sigma$  and their corresponding formulas over  $\text{rec } t.\Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes t)$ , each of which implicitly has an  $\text{inl}$  prefix.

Strings can then be encoded accordingly in an unambiguous way. For example, the string  $abab$  corresponds to the formula  $\text{inr}(a \cdot (\text{inr}(b \cdot (\text{inr}(a \cdot (\text{inl } b))))))$ . The type inference that led to this is the following. Since  $b$  is a formula of the type  $\Sigma_{\perp}$ ,  $\text{inl } b$  is a formula of  $\Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes t)$ , and hence a formula of  $\text{rec } t.\Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes t)$ . One can then repeatedly use  $\cdot$  and  $\text{inr}$  to obtain more formulas of this type.

In general, a terminating  $\mu$ -formula of type  $P$  determines a regular language by the following inductive definition:

- $\mathcal{R}(f) = \emptyset$ .
- $\mathcal{R}(\text{inl } a) = \{a\}$  for each  $a \in \Sigma$  (see the encoding of alphabet above).
- $\wedge$  corresponds to intersection and  $\vee$  corresponds to union.
- if  $\mathcal{R}(\varphi) = A$ , and  $a \in \Sigma$ , then  $\mathcal{R}(\text{inr}(a \cdot \varphi)) = \{aw \mid w \in A\}$ .
- $\mathcal{R}(\mu x.\varphi(x)) = \bigcup_{i \geq 0} \mathcal{R}(\varphi^i(f))$ .

The main result of this section is the following.

- Theorem 4.1.** (1) For each terminating formula  $\varphi$ ,  $\mathcal{R}(\varphi)$  is an  $\epsilon$ -free regular language, and for every such language  $L$ , there exists a terminating formula  $\varphi$  such that  $\mathcal{R}(\varphi) = L$ .
- (2) There is an effective procedure to find the regular language determined by a terminating formula.
- (3) For terminating formulas  $\varphi, \psi$ ,  $\llbracket \varphi \rrbracket \subseteq \llbracket \psi \rrbracket$  if and only if  $\mathcal{R}(\varphi) \subseteq \mathcal{R}(\psi)$ , and the problems of semantic containment  $\llbracket \varphi \rrbracket \subseteq \llbracket \psi \rrbracket$  and emptiness  $\llbracket \varphi \rrbracket = \emptyset$  are decidable.

Note that nonterminating formulas are those which are semantically equivalent to  $t$ . The correspondence has to leave out languages containing the empty string  $\epsilon$ , as it cannot be expressed by a  $\mu$ -formula. On the other hand, the formula  $t$  represents the whole domain  $D_{\text{rec } t.\Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes t)}$ , which does not correspond to a language due to the presence of bottom element.

**Example.** For  $\Sigma = \{0, 1\}$ , the formulas  $(\text{inr}(0 \cdot \text{inl } 1)) \vee (\text{inr}(1 \cdot \text{inl } 0))$ ,  $\mu x.(\text{inl } 0 \vee \text{inr}(\text{inl } 0 \cdot x))$ , and  $\mu x.(\text{inl } 0 \vee \text{inl } 1) \vee (\text{inr}((\text{inl } 0 \vee \text{inl } 1) \cdot x))$  determine the regular expressions  $01 + 10$ ,  $0^+$ , and  $(0 + 1)^+$ , respectively.

*Proof of Theorem 4.1.* We prove item 2 first. Let  $\varphi$  be a (closed) terminating formula. First rename all of its bound variables so that they are all distinct from each other. We use the standard notion of *sub-formulas* for  $\mu$ -formulas to derive a system of language equations associated with  $\varphi$ .

- The only sub-formulas of **t**, **f** and variables are the formulas themselves.
- The sub-formulas of  $\varphi \wedge \psi$  and  $\varphi \vee \psi$  consist of the formulas themselves together with sub-formulas of  $\varphi$  and  $\psi$ .
- The sub-formulas of  $\text{inr}(a \cdot \varphi)$  consist of the formula itself,  $a$ , and the sub-formulas of  $\varphi$ .
- The sub-formulas of  $\mu x. \varphi(x)$  consist of the formula itself and sub-formulas of  $\varphi(x)$ .

Now, for each sub-formula  $\psi$  of  $\varphi$ , introduce a distinct variable  $X_\psi$  associated with it. Then build a system of equations as follows (by using  $\mathcal{R}$  to encode propositional formulas):

- If  $p$  is a closed, propositional sub-formula of  $\varphi$ , then add equation  $X_p = p$ .
- If  $\psi_1 \wedge \psi_2$  is a sub-formula of  $\varphi$ , then add equation  $X_{\psi_1 \wedge \psi_2} = X_{\psi_1} \wedge X_{\psi_2}$ . Similarly for  $\vee$ .
- If  $\psi \equiv \text{inr}(a \cdot \varphi)$  is a sub-formula of  $\varphi$ , then add equation  $X_\psi = a \cdot X_\varphi$ .
- Finally, if  $\mu z. \psi$  is a sub-formulas of  $\mu x. \varphi(x)$ , then add equation  $Z = X_\psi$ .

For example, the formula  $\mu x.(\text{inl } 0 \vee \text{inr}(\text{inl } 0 \cdot x))$  determines the following system of equations:

$$\begin{cases} X_0 = 0 \\ Y = 0 \cdot X \\ Z = X_0 \vee Y \\ X = Z \end{cases}$$

One can check that this equation system has the  $\epsilon$ -property (Definition 3.4), and, it therefore has a unique solution in regular languages which can be constructed using an effective procedure (Theorem 3.7).

We need to show next that every terminating formula gives rise to a system of language equations with the  $\epsilon$ -property. But this is not true in general. For example, the equation associated with  $\mu x. x$  is  $X = X$ , which does not have the  $\epsilon$ -property (although it does have a solution). A closer inspection reveals that such a situation arises only when a variable  $x$  in a  $\mu$ -formula  $\mu x. \varphi(x)$  does not occur in a context of the form  $\text{inr}(a \cdot \psi(x))$ . In such a situation, we can replace such  $\mu$ -formulas by starting with the innermost sub-formula  $\mu x. \varphi(x)$ , converting  $\varphi(x)$  to a disjunctive normal form, and substituting  $\mu x. \varphi(x)$  by a formula obtained by replacing all the conjuncts in which  $x$  occurs by **f**, then repeat the procedure again until no such variable occurs. This way, we obtain a system of equations with the  $\epsilon$ -property whose solution provides the semantics of a given  $\mu$ -formula.

To see that the resulting solution of the system of equations captures the same semantics as the original formula, it suffices to show that with respect to  $\mu x. \varphi(x)$ , the language  $\bigcup_{i \geq 0} \mathcal{R}(\varphi^i(\mathbf{f}))$  is a part of the solution to the system of language equations associated with  $\mu x. \varphi(x)$ , for the equality  $X = X_{\varphi(x)}$ . The uniqueness of solution (Theorem 3.7) then ensues that  $\mathcal{R}(\mu x. \varphi(x))$  is a regular language.

The fact that  $\bigcup_{i \geq 0} \mathcal{R}(\varphi^i(\mathbf{f}))$  is indeed part of the desired solution comes from a fixed-point reformulation of systems of language equations. A system of language equations

$$\begin{cases} X_1 = \varphi_1(X_1, \dots, X_n) \\ \dots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{cases}$$

as given in Definition 3.3 determines a function  $\Phi : (2^{\Sigma^*})^n \rightarrow (2^{\Sigma^*})^n$  with

$$\Phi(L_1, \dots, L_n) \stackrel{\text{def}}{=} (\varphi_1(L_1, \dots, L_n), \dots, \varphi_n(L_1, \dots, L_n))$$

for each language vector  $(L_1, \dots, L_n) \in (2^{\Sigma^*})^n$ . Under coordinatewise set-inclusion,  $(2^{\Sigma^*})^n$  is a complete lattice, and  $\Phi$  is a continuous function in the standard domain-theoretic sense, since the operators of union, intersection, and concatenation are monotonic and continuous (note that negation is not an operator considered here). By the continuity of  $\Phi$ , the fixed-point theorem in domain theory provides the least fixed-point  $\mathbf{L}$  of  $\Phi$ , with  $\mathbf{L} = \bigcup_{i \geq 0} \Phi^i(\emptyset, \dots, \emptyset)$ . This implies that the solution for  $X = X_{\varphi(x)}$  takes the form  $\bigcup_{i \geq 0} \mathcal{R}(\varphi^i(\mathbf{f}))$ , by substituting equals by equals.

To prove item 1, it suffices to show that each  $\epsilon$ -free regular language over  $\Sigma$  can be represented by a  $\mu$ -formula of  $P$ , because item 2 covers the other direction of item 1. Let  $L$  be an  $\epsilon$ -free regular language over  $\Sigma$ . By standard results of automata theory, there exists a DFA accepting  $L$ , with  $n$  states  $X_1, \dots, X_n$ , where the initial state  $X_1$  is not a final state (otherwise it would have accepted the empty string  $\epsilon$ ). Such a DFA can be represented as an equation system

$$\begin{cases} X_1 = \varphi_1(X_1, \dots, X_n) \\ \dots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{cases}$$

where each  $\varphi_i(X_1, \dots, X_n)$  takes a simple linear form

$$a_1 \cdot X_{i_1} \vee a_2 \cdot X_{i_2} \vee \dots \vee a_m \cdot X_{i_m} \vee L_i,$$

with  $\{a_i \mid i = 1, \dots, m\} = \Sigma$  and  $L_i = \{\epsilon\}$  if  $X_i$  is a final state, and  $L_i = \emptyset$  otherwise. We use the idea of ‘‘Gaussian-elimination’’ to obtain the desired  $\mu$ -formula. Replacing all occurrences of  $X_1$  by the formula  $\xi_1(X_2, \dots, X_n) \equiv \mu X_1. \varphi_1(X_1, \dots, X_n)$  in equations 2 to  $n$ , and we obtain a system of equations in

$$(X_2, \dots, X_n): \begin{cases} X_2 = \varphi_2(\xi_1(X_2, X_3, \dots, X_n), \dots, X_n) \\ \dots \\ X_n = \varphi_n(\xi_1(X_2, X_3, \dots, X_n), \dots, X_n) \end{cases}$$

Next, replace all occurrences of  $X_2$  by the formula

$$\xi_2(X_3, \dots, X_n) \equiv \mu X_2. \varphi_2(\xi_1(X_2, X_3, \dots, X_n), \dots, X_n)$$

in equations 3 to  $n$  to obtain

$$\begin{cases} X_3 = \varphi_3(\xi_1(\xi_2, X_3, \dots, X_n), \xi_2, \dots, X_n) \\ \dots \\ X_n = \varphi_n(\xi_1(\xi_2, X_3, \dots, X_n), \xi_2, \dots, X_n) \end{cases}$$

In  $n$  steps, we obtain  $\xi_n \equiv \mu X_n. \varphi'(X_n)$  where  $\varphi'(X_n)$  is a formula without variables other than  $X_n$ ; so  $\xi_n$  is closed. Now start substituting backwards, to obtain a closed formula for  $\xi_{n-1}$ , and then  $\xi_{n-2}$ , etc., until we obtain a closed formula for  $\xi_1$ . Now  $\xi_1$  would be the  $\mu$ -formula to encode the language  $L$  except for one problem: the presences of  $\epsilon$  in some places for which we have no corresponding formula to represent. However, this problem can be overcome by noting that the linear form for  $X_1$  does not involve  $\epsilon$ , and  $a \cdot (\mu X. \varphi(X))$  is a formula equivalent in meaning to  $\mu X. (a \cdot \varphi(X))$ . The  $\epsilon$ -term in  $\varphi(X)$  can then be absorbed by using the distributivity of  $\cdot$  over  $\vee$  recursively, since  $\varphi$  is a linear term.

To see that the procedure is correct, i.e. the resulting  $\mu$ -formula indeed represents the solutions, notice that the  $\mu$ -formulas can be understood as a syntactic coding of solutions to their associated equations. A  $\mu$ -formula of the form  $\mu x. \varphi(x)$  stands for the least fixed point of  $\varphi$ , so we have  $\varphi(\mu x. \varphi(x)) = \mu x. \varphi(x)$  both semantically and proof-theoretically.

What remains to be shown is that for terminating formulas  $\varphi, \psi$ ,  $\llbracket \varphi \rrbracket \subseteq \llbracket \psi \rrbracket$  if and only if  $\mathcal{R}(\varphi) \subseteq \mathcal{R}(\psi)$ . This is because the inductive definitions for  $\llbracket \cdot \rrbracket$  and  $\mathcal{R}$  take precisely the same form.  $\square$

We provide an example to show how to derive a  $\mu$ -formula for an  $\epsilon$ -free language accepted by some DFA, using the ‘‘Gaussian-elimination’’ idea described in the proof of Theorem 4.1.

**Example.** Consider the following language equations corresponding to a DFA accepting some  $\epsilon$ -free language (where  $X_1$  represents the initial state):

$$\begin{cases} X_1 = aX_2 \vee bX_1 \\ X_2 = aX_2 \vee bX_3 \vee \epsilon \\ X_3 = bX_3 \vee aX_1 \vee \epsilon \end{cases}$$

First, we eliminate  $X_1$  by replacing all of its occurrences in the 2nd and 3rd equation by  $\mu X_1. (aX_2 \vee bX_1)$ :

$$\begin{cases} X_2 = aX_2 \vee bX_3 \vee \epsilon \\ X_3 = bX_3 \vee a(\mu X_1. (aX_2 \vee bX_1)) \vee \epsilon \end{cases}$$

Next, we eliminate  $X_2$  by replacing all of its occurrences in the 3rd equation by  $\mu X_2. (aX_2 \vee bX_3 \vee \epsilon)$ :

$$X_3 = bX_3 \vee a(\mu X_1. (a(\mu X_2. (aX_2 \vee bX_3 \vee \epsilon)) \vee bX_1)) \vee \epsilon$$

Then  $X_3$  is replaced by the closed formula

$$\mu X_3. (bX_3 \vee a(\mu X_1. (a(\mu X_2. (aX_2 \vee bX_3 \vee \epsilon)) \vee bX_1)) \vee \epsilon)$$

Substituting  $X_3$  by this formula we obtain, for  $X_2$ ,

$$\begin{aligned} & \mu X_2. (aX_2 \vee bX_3 \vee \epsilon) \\ &= \mu X_2. \epsilon \vee aX_2 \vee \\ & \quad b(\mu X_3. \epsilon \vee bX_3 \vee \\ & \quad \quad a(\mu X_1. bX_1 \vee \\ & \quad \quad \quad a(\mu X_2. \epsilon \vee aX_2 \vee bX_3))) \end{aligned}$$

By one more substitution, we obtain a closed formula for  $X_1$ :

$$\begin{aligned} & \mu X_1. bX_1 \vee \\ & \quad a(\mu X_2. \epsilon \vee aX_2 \vee \\ & \quad \quad b(\mu X_3. \epsilon \vee bX_3 \vee \\ & \quad \quad \quad a(\mu X_1. bX_1 \vee \\ & \quad \quad \quad \quad a(\mu X_2. \epsilon \vee aX_2 \vee bX_3)))) \end{aligned}$$

Finally, the occurrences of  $\epsilon$ 's are absorbed by the distributive law ( $\cdot$  over  $\vee$ ), where some bound variables have been renamed for clarity:

$$\begin{aligned} & \mu X_1. bX_1 \vee \\ & \quad \mu X_2. a \vee aX_2 \vee \\ & \quad \quad a(\mu X_3. b \vee bX_3 \vee \\ & \quad \quad \quad ba(\mu Y_1. bY_1 \vee \\ & \quad \quad \quad \quad (\mu Y_2. a \vee aY_2 \vee abX_3))) \end{aligned}$$

This can now be easily translated to a  $\mu$ -formula for the given DFA.

## 5. $\mu$ -CALCULUS FOR $Q = \Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes Q \otimes \Sigma_{\perp})$

If the  $\mu$ -formulas of  $\text{rec } t. \Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes t)$  correspond to regular languages, what language class do the formulas of type  $\text{rec } t. \Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes t \otimes \Sigma_{\perp})$  (and  $\text{rec } t. \Sigma_{\perp} \oplus (t \otimes t)$ ) correspond to? Note that this type is chosen to make it seemingly possible to simulate context-free rewriting in the form of  $A \rightarrow w_1 B w_2$ , as used for context-free languages. As emptiness and containment for context-free languages are well-known to be *undecidable*, one would guess that this fragment of  $\mu$ -formulas is undecidable.

In formal language theory, the essential feature of the concatenation operation for strings is that it is associative:  $(ab)c = a(bc) = abc$ . However, one crucial fact about the  $\mu$ -calculus makes the correspondence unfeasible: the smash product construction is not associative. As pointed out at the end of Section 1.3, we do not in general have  $\varphi_1 \cdot (\varphi_2 \cdot \varphi_3) = (\varphi_1 \cdot \varphi_2) \cdot \varphi_3$ . This is also reflected in SML's datatype construction. For example, if one defines

$$\text{datatype } P = L \mid C \text{ of } P * P$$

then testing for equality of  $C(L, C(L, L))$  and  $C(C(L, L), L)$  yields **false**. More to the point – as pointed out in the conclusion section – it is not known if the  $\mu$ -calculus for  $\text{rec } t. \Sigma_{\perp} \oplus (t \otimes t)$  is decidable or not. If associativity were to hold, this would have allowed us to encode all context-free languages, in the standard Chomsky normal form (using production rules  $A \rightarrow w$  and  $A \rightarrow BC$ ), for which containment is undecidable.

The main result of this section is that the problems of semantic entailment (is it true that  $\llbracket \varphi \rrbracket \subseteq \llbracket \psi \rrbracket$ ?) and emptiness (is it true that  $\llbracket \varphi \rrbracket = \emptyset$ ?) for the  $\mu$ -calculus for  $\text{rec } t. \Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes t \otimes \Sigma_{\perp})$  is *decidable*. Some very early results on *even linear* languages [5, 6] were found to be helpful by serendipity. These are briefly recalled next, in modern terminology, to make the paper self-contained.

### 5.1. EVEN LINEAR LANGUAGES

**Definition 5.1.** An even linear grammar is a context-free grammar  $(\Sigma, S, N, P)$  where  $\Sigma$  is a finite set of terminal symbols (the alphabet),  $N$  a finite set of non-terminal symbols,  $S \in N$  the start symbol, and  $P$  a set of production rules of the form

$$A \rightarrow w_1 \text{ or } A \rightarrow w_2 B w_3, \text{ with } |w_2| = |w_3|,$$

where  $A, B \in N$ , and  $w_i \in \Sigma^*$  for  $i = 1, 2, 3$ . Equivalently, one can restrict production rules to the form

$$A \rightarrow w_1 \text{ or } A \rightarrow w_2 B w_3, \text{ with } |w_2| = |w_3|, |w_i| \leq 1$$

without losing expressive power. A language is called even linear if it can be generated by some even linear grammar.

**Example.** It is easy to see that the language  $\{a^n b a^n \mid n \geq 1\}$  is even linear but not regular.

The following is the basic classification result about even linear languages.

**Theorem 5.2** (Amar and Putzolu). *The class of even linear languages strictly contains the class of regular languages.*

Although the proof is non-trivial, it is not recalled here because our focus will be on decidability and closure properties: are the emptiness and containment problems for even linear languages decidable? What kind of language operations preserve even linear languages?

## 5.2. BALANCED TWO-WAY AUTOMATA, MINIMIZATION, AND DECIDABILITY

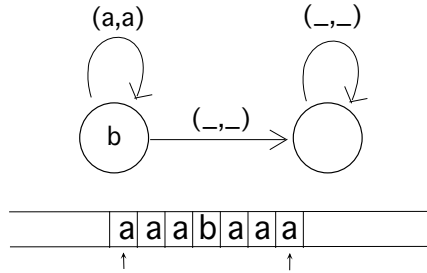
The decidability of semantic entailment and emptiness for the  $\mu$ -calculus of type  $Q$  (not to be confused with the state set  $Q$  below) will be answered in the affirmative by using an idea from regular languages.

**Definition 5.3.** A balanced two-way deterministic finite automaton (DFA)  $M$  is a 5-tuple  $(\Sigma, Q, \delta, s, F)$ , where

- $\Sigma, Q, s$  are the alphabet set, state set, and starting state, as before;
- $\delta : (\Sigma \times \Sigma) \times Q \rightarrow Q$  the transition function, taking a pair of symbols at a time, and
- $F : Q \rightarrow (\Sigma \cup \{\epsilon\}) \cup \{\emptyset\}$  is the set of final states together with the accepting symbol. A state  $q$  is not final if  $F(q) = \emptyset$ .

Given an input string, a balanced two-way DFA starts from the starting state with the two scanner heads resting on each end of the string. Both heads then move inwards in a synchronous fashion one symbol at a time, and stop when the number of input symbols remaining to be scanned is less or equal to one. The input string is accepted if the DFA stops at a final state and the remaining symbol, if any, matches the symbolic label of the state. Otherwise the string is rejected.

**Example.** Here is the picture of a balanced two-way DFA accepting the language  $\{a^n b a^n \mid n \geq 1\}$ .



The next result follows from standard conversion of a nondeterministic finite state machine to a deterministic one.

**Theorem 5.4.** *A language is even linear if and only if it is accepted by some balanced two-way DFA.*

A notion of “right” invariant relation can be introduced to obtain canonical forms of balanced two-way DFA.

**Definition 5.5.** Let  $L \subseteq \Sigma^*$ . For  $u, v \in \Sigma^*$ , define

$$u R_L v \stackrel{\text{def}}{\Leftrightarrow} \forall x, y \text{ with } |x| = |y|, xuy \in L \text{ iff } xvy \in L.$$

The relation  $R_L$  has these properties.

- For any  $L$ ,  $R_L$  is an equivalence relation.
- If  $u R_L v$  then  $aub R_L avb$  for any  $a, b \in \Sigma$ .
- $R_L$  is saturated:  $L = \bigcup_{u \in L} [u]_{R_L}$ , where  $[u]_{R_L} := \{v \mid u R_L v\}$ .
- $L$  is even linear if and only if  $R_L$  has finite index (i.e., finite number of equivalence classes).

Based on these, one can define two states  $p, q$  in a balanced two-way DFA to be equivalent if  $F(\delta^*(p, w)) = F(\delta^*(q, w))$  for any sequence of symbol pairs  $w = (a_1, b_1)(a_2, b_2) \cdots (a_n, b_n)$ , with  $a_i, b_i \in \Sigma$  for each  $1 \leq i \leq n$ . Using standard ideas from automata theory, we can then obtain an algorithm which minimizes the number of states of a balanced two-way DFA in a canonical way. Thus, we have

**Theorem 5.6.** *Emptiness and containment of even linear languages are decidable.*

Moreover, the family of even linear languages is the least family of languages containing finite languages and closed under  $+$ ,  $\odot$ ,  $\star$ , where  $+$  is union,  $\star$  is repetition of  $\odot$ , where  $\odot$  is defined in the non-standard way – for example, at the string level, we have

$$abc \odot aaabbbccc := aaababc bccc$$

In general,  $u \odot v$  is defined as the string by wrapping the first half of  $v$  in front of  $u$  and the second half of  $v$  in the end of  $u$  (in the case that  $v$  is uneven, the middle symbol is ignored). This gives a Kleene algebra (in the sense of Kozen [21]; see [7] as well) with a non-standard interpretation. An intuitive way to understand what's going on here is to regard this non-standard Kleene algebra as a standard one over the new alphabet  $\Sigma \times \Sigma$ , instead of  $\Sigma$ . For example, each even linear grammar over  $\{a, b\}$  corresponds to a triple  $(G, G_a, G_b)$  of regular grammars over the alphabet  $\Sigma \times \Sigma$ . The regular grammars are obtained by first translating each production of the form  $A \rightarrow xBy$  with  $x, y \in \{a, b\}$  to a production of the form  $A \rightarrow (x, y)B$ , with  $(x, y)$  a member of the new alphabet  $\{a, b\}^2$ . Then  $G$  consists of the derived productions  $A \rightarrow (x, y)B$  together with the production  $A \rightarrow \epsilon$ , if it is in the original production set.  $G_a$  consists of the derived productions  $A \rightarrow (x, y)B$  together with the production  $A \rightarrow \epsilon$ , provided that  $A \rightarrow a$  is in the original production set, and similarly for  $G_b$ .

Just as in classical automata theory, the interplay among balanced two-way finite state machines, even-linear grammars, and regular expressions provides insight about three distinct aspects of the class of even linear languages: combinatorial, algebraic, and equational. In particular, the automata-theoretic account of even linear languages offers the method to show that this class of languages is closed under union, intersection, Kleene star, and complement. (The contribution of Amar and Putzolu is that this class of languages, regular over  $\Sigma \times \Sigma$  by encoding, contains all regular languages over  $\Sigma$ .)

### 5.3. DECIDABILITY OF THE $\mu$ -CALCULUS FOR $\text{rect} t.\Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes t \otimes \Sigma_{\perp})$

After these preparations, we are ready to state the main result of the section.

**Theorem 5.7.** *The emptiness problem and the containment problem for the  $\mu$ -calculus for  $\text{rec } t.\Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes t \otimes \Sigma_{\perp})$  is decidable. The  $\mu$ -formulas express even linear languages of odd-length strings over  $\Sigma$ .*

*Proof.* The proof follows similar steps as that of Theorem 4.1. We therefore outline the major steps but leaves out the details. First, encode each  $\mu$ -formula as an even linear grammar over  $\Sigma$ . Then encode the even linear grammar as a system of language equations over the alphabet  $\Sigma \times \Sigma$  (using the same idea explained near the end of the last subsection). The change of alphabet is to make sure that we obtain language equations conforming to Definition 3.3, for which only left-concatenation is permitted. We then follow exactly the same strategy in the proof of Theorem 4.1 to derive the decidability result and obtain a characterization of the expressive power of the domain  $\mu$ -calculus for  $\text{rec } t.\Sigma_{\perp} \oplus (\Sigma_{\perp} \otimes t \otimes \Sigma_{\perp})$ .  $\square$

## 6. CONCLUDING REMARKS

We have formulated a basic logical framework for the domain  $\mu$ -calculus and proved decidability and expressiveness results for several non-trivial fragments of the  $\mu$ -calculus. Some results in automata theory and language equations are employed for our study.

One can use a notion of *proportional* linear grammar [5] to provide decidability results for a larger fragment of domain  $\mu$ -calculus, with small variations on the type definitions. Although similar ideas and techniques may work for a bigger segment, our results have not achieved full generality. For example, it is not clear how to treat the “non-linear” fragments, such as  $\text{rec } t.\Sigma_{\perp} \oplus (t \otimes t)$ . The decidability problem for the general  $\mu$ -calculus remains open – an undecidable fragment is yet to be found.

Reduction to tree languages of certain kind seems to be a plausible way to perhaps completely resolve the decidability issue. However, we feel that the current restricted sense of achievement is due in large part to the inherent combinatorial nature of the problem. Completeness of the whole domain  $\mu$ -calculus is expected to be harder.

Other type-constructions such as function space and powerdomains can be brought into the picture. However, unrestricted use of function space necessarily carries us outside the realm of Scott open sets (since  $(\bigcup A_i) \rightarrow B = \bigcap (A_i \rightarrow B)$ ). It is not clear what the larger topological space would be, if other than Scott. However, [8] is a good start in this direction.

Also note that language equations of a more general format has been studied under the name of *conjunctive grammars* [27, 28], which might be helpful in the study of non-linear fragments of the  $\mu$ -calculus such as the one for  $\text{rec } t.\Sigma_{\perp} \oplus (t \otimes t)$ .

**Acknowledgment** I would like to thank Ernst Leiss and Alexander Okhotin for discussions related to Boolean automata and language equations. The anonymous referees provided suggestions and comments which prompted me to look deeper into some overlooked aspects in the earlier version. I am grateful for their feedback.

## REFERENCES

- [1] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, Vol. 51, pp. 1-77, 1991.
- [2] S. Abramsky and A. Jung, Domain theory, in: *Handbook of Logic in Computer Science*, Vol. 3, pp. 1-168, Clarendon Press, 1995.
- [3] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, Vol. 92, pp. 161-218, 1991.
- [4] A. Arnold. The mu-calculus alternation-depth hierarchy is strict on binary trees. *R.A.I.R.O. Informatique Théorique et Applications*, Vol. 33, pp. 329-339, 1999.
- [5] V. Amar and G. Putzolu. On a family of linear grammars. *Information and Control*, Vol. 7, pp. 283-291, 1964.
- [6] V. Amar and G. Putzolu. Generalizations of regular events. *Information and Control*, Vol. 8, pp. 56-63, 1965.
- [7] S. Bloom and Z. Ésik. Equational axioms for regular sets. Technical Report 9101, Stevens Institute of Technology, 1991.
- [8] M. Bonsangue and J. N. Kok. Towards an infinitary logic of domains: Abramsky logic for transition systems. *Information and Computation*, Vol. 155, pp. 170-201, 1999.
- [9] J.C. Bradfield. Simplifying the modal mu-calculus alternation hierarchy. *Lecture Notes in Computer Science*, Vol. 1373, pp. 39-49, 1998.
- [10] S. Brookes. A semantically based proof system for partial correctness and deadlock in CSP. In Proceedings, *Symposium on Logic in Computer Science*, pp. 58-65, Cambridge, Massachusetts, 1986.
- [11] J. Brzozowski and E. Leiss. On equations for regular languages, finite automata, and sequential networks. *Theoretical Computer Science*, Vol. 10, pp. 19-35, 1980.
- [12] A.K. Chandra, D. Kozen, and L. Stockmyer. Alternation. *Journal of the ACM*, Vol. 28, pp. 114-133, 1981.
- [13] Z. Ésik. Completeness of Park induction. *Theoretical Computer Science*, Vol. 177, pp. 217 - 283, 1997 (MFPS'94).
- [14] A. Fellah. Alternating finite automata and related problems. PhD Thesis, Department of Mathematics and Computer Science, Kent State University, 1991.
- [15] A. Fellah, H Jurgensen, S. Yu. Constructions for alternating finite automata. *International Journal of Computer and Mathematics*, Vol. 35, pp. 117-132, 1990.
- [16] C. Gunter and D. Scott. Semantic domains. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, Vol. B, pp 633-674, Elsevier, 1990.
- [17] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. *Lecture Notes in Computer Science (CONCUR'96)*, Vol. 1119, pp. 263-277, 1996.
- [18] T. Jensen. Disjunctive program analysis for algebraic data types. *ACM Transactions on Programming Languages and Systems*, Vol. 19, pp. 752-804, 1997.
- [19] P.T. Johnstone. *Stone Spaces*. Cambridge University Press, 1982.
- [20] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, Vol. 27, pp. 333-354, 1983.
- [21] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, Vol. 110, pp. 366-390, 1994.
- [22] E. Leiss. Succinct representation of regular languages by Boolean automata. *Theoretical Computer Science*, Vol. 13, pp. 323-330, 1981.
- [23] E. Leiss. *Language Equations*. Monographs in Computer Science, Springer-Verlag, New York, 1999.
- [24] R.S. Lubarsky.  $\mu$ -definable sets of integers. *Journal of Symbolic Logic*, Vol. 58, pp. 291-313, 1993.
- [25] D. Niwiński. Fixed points vs. infinite generation. *Logic in Computer Science (LICS'88)*, pp. 402-409, IEEE Computer Press, 1988.

- [26] D. Niwiński. Fixed point characterization of infinite behaviour of finite state systems. *Theoretical Computer Science*, Vol. 189, pp. 1-69, 1997.
- [27] A. Okhotin. Automaton representation of linear conjunctive languages. In Proceedings of DLT 2002, *Lecture Notes in Computer Science*, to appear.
- [28] A. Okhotin. On the closure properties of linear conjunctive languages. *Theoretical Computer Science*, to appear.
- [29] D. Park. Concurrency and automata on infinite sequences. *Lecture Notes in Computer Science*, Vol. 154, pp. 561-572, 1981.
- [30] G. Plotkin. *The Pisa Notes*. Department of Computer Science, University of Edinburgh, 1981.
- [31] G. Plotkin. A powerdomain construction. *SIAM J. Computing*, Vol. 5, pp. 452-487, 1976.
- [32] V. R. Pratt. A decidable mu-calculus: Preliminary report. In Proceedings of *IEEE 22nd Annual Symposium on Foundations of Computer Science*, pp. 421-427, 1981.
- [33] M. Presburger. On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operator. *Hist. Philos. Logic*, 12:225-233, 1991 (English translation of the original paper in 1930).
- [34] M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res.* Vol. 3, pp. 115-125, 1959.
- [35] M. Y. Vardi. Alternating automata and program verification. In *Computer Science Today – Recent Trends and Developments, Lecture Notes in Computer Science* Vol. 1000, pp. 471-485, 1995.
- [36] I. Walukiewicz. Completeness of Kozen’s axiomatisation of the propositional  $\mu$ -calculus. *Information and Computation*, Vol. 157, pp. 142-182, 2000.
- [37] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.
- [38] S. Yu. Regular Languages. In *Handbook of Formal Languages*, Rozenberg and Salomaa (eds.), Springer-Verlag, pp. 41-110, 1997.
- [39] G.-Q. Zhang. *Logic of Domains*. Birkhauser, Boston, 1991.

Communicated by (The editor will be set by the publisher).  
August 18, 2003.