On the Hoare Theory of Monadic Recursion Schemes

Konstantinos Mamouras
Cornell University
mamouras@cs.cornell.edu

Abstract

The equational theory of monadic recursion schemes is known to be decidable by the result of Sénizergues on the decidability of the problem of DPDA equivalence. In order to capture some properties of the domain of computation, we augment equations with certain hypotheses. This preserves the decidability of the theory, which we call *simple implicational theory*. The asymptotically fastest algorithm known for deciding the equational theory, and also for deciding the simple implicational theory, has running time that is non-elementary. We therefore consider a restriction of the properties about schemes to check: instead of arbitrary equations $f \equiv g$ between schemes, we focus on propositional Hoare assertions $\{p\}f\{q\}$, where f is a scheme and p, q are tests. Such Hoare assertions have a straightforward encoding as equations. We investigate the *Hoare theory* of monadic recursion schemes, that is, the set of valid implications whose conclusions are Hoare assertions and whose premises are of a certain simple form. We present a sound and complete Hoare-style calculus for this theory. We also show that the Hoare theory can be decided in exponential time, and that it is complete for this class.

Categories and Subject Descriptors F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs—Logics of programs; F.3.3 [*Logics and Meanings of Programs*]: Studies of Program Constructs—Program and recursion schemes

General Terms Theory, Verification, Languages

Keywords Hoare logic, propositional Hoare logic, monadic recursion schemes, monadic program schemes, sound and complete Hoare calculus

1. Introduction

The equivalence problem for pushdown automata (PDAs) is a standard undecidable problem. In fact, it is Π_1^0-complete and therefore not even recursively enumerable. For a special subclass of PDAs, called deterministic PDAs or DPDAs, the question of the decidability of language-equivalence was posed in [9]. After remaining open for three decades, this question was settled positively by Sénizergues in [20] (journal version [21]). Simplified proofs of this decidability result were presented later by Stirling [24] and Sénizergues [22]. Stirling has also obtained a primitive recursive upper bound for the problem [25], but the proposed algorithm witnessing this bound has worst-case running time that is non-elementary. Recently, Jančar gave a simplified proof of the decidability result for DPDA equivalence [11, 12]. Jančar's proof relies on the use of first-order terms and grammars.

DPDA equivalence is related to the problem of equivalence of monadic recursive program schemes (also called monadic recursion schemes). The atomic actions and predicates in such schemes are uninterpreted, and hence completely abstract. The schemes are called monadic because they only have one variable. In other words, the entire state of the program is viewed as an indivisible entity, as opposed to the case of being able to "see" various variables that can be set and read separately. The (strong) equivalence problem for program schemes is checking whether two schemes denote the same partial function under every possible interpretation of the atomic actions and predicates. Garland and Luckham showed in [8] (page 132, Theorem 2.10, part (b)) that the equivalence of monadic recursion schemes can be reduced to the equivalence problem for deterministic context-free grammars (these grammars correspond to DPDAs). Moreover, Friedman showed in [7] the converse, namely, that DPDA equivalence can be reduced to monadic recursion scheme equivalence. So, by the results of Sénizergues and Stirling, the *equational theory* of monadic recursion schemes (the set of equations between such schemes that hold under every interpretation) is decidable and, in fact, is a primitive recursive set.

The decidability of the equational theory of monadic recursion schemes suggests that this formalism can offer a convenient level of abstraction at which to reason about the control structure of recursive deterministic programs. However, in order to use such schemes for real programming applications, we need to reason under hypotheses that capture some properties of the domain of computation. For example, consider the following equivalent programs:

program 1	program 2
$x := 1;$	$x := 1;$
$\mathsf{if}\ (x \geq 0)\ \mathsf{then}\ y := 2\ \mathsf{else}\ y := 3$	$y := 2$

The equivalence of the above programs hinges on a property of the domain of computation (the integers) that can be expressed with the Hoare assertion $\{\mathsf{true}\}x := 1\{x \geq 0\}$. This assertion is read as follows: "after the execution of the statement $x := 1$, the test $x \geq 0$ holds". However, for the monadic schematic abstractions

$$a; \mathsf{if}\ p\ \mathsf{then}\ b\ \mathsf{else}\ c \quad \text{and} \quad a; b$$

of the above programs (where a, b, c are abstract atomic actions replacing the statements $x := 1$, $y := 2$, and $y := 3$ respectively, and p is an abstract atomic test replacing $x \geq 0$) equivalence does not hold. Now, reasoning under the hypothesis $\{\mathsf{true}\}a\{p\}$, the schemes $a; \mathsf{if}\ p\ \mathsf{then}\ b\ \mathsf{else}\ c$ and $a; b$ can be shown to be equivalent. This simple example suggests that it would be desirable to be able to handle implications, e.g.

$$\{\mathsf{true}\}a\{p\} \Rightarrow a; b \equiv a; \mathsf{if}\ p\ \mathsf{then}\ b\ \mathsf{else}\ c,$$

CSL-LICS 2014, July 14–18, 2014, Vienna, Austria.

http://dx.doi.org/10.1145/2603088.2603157

in addition to just equations. If the hypotheses are allowed to be arbitrary equations, the theory is rendered undecidable [19] (see also [6]). So, we are led to consider here more restricted hypotheses that are either Hoare assertions for atomic actions (that is, statements of the form $\{p\}a\{q\}$), or propositional formulas for tests. The set of valid implications $\Phi \Rightarrow f \equiv g$, where Φ is a collection of thus restricted hypotheses, is called the *simple implicational theory* of monadic recursion schemes.

The best known algorithm for deciding DPDA equivalence, due to Stirling [25], has non-elementary asymptotic running time. As far as the inherent computational complexity of the problem is concerned, no non-trivial lower bounds are known. The complexity gap between the known P-hard lower bound and the primitive recursive upper bound has motivated the study of further subclasses of DPDAs. Sénizergues shows in [23] that for every integer $t \geq 1$, the equivalence problem for t-turn DPDAs lies in coNP. In such DPDAs the number of switches between pushing to and popping from the stack is bounded. Böhm, Göller, and Jančar study deterministic one-counter automata, which extend the standard DFAs with a non-negative counter, and show that their equivalence problem is NL-complete [2, 3]. An earlier result for deterministic real-time one-counter automata was obtained in [1].

For the works mentioned in the previous paragraph, the decision problem of scheme equivalence was shown to be easier by restricting the functionality of the stack of the DPDA. Intuitively, this can be understood in the context of program schemes as restricting recursion. In the present work we explore a different way of obtaining an easier decision problem: we do not restrict recursion, but rather we check a property that is simpler than equivalence. For an arbitrary monadic recursion scheme f, we check a property given by the Hoare assertion $\{p\}f\{q\}$. This assertion expresses the same property as the equation

$$\bot \equiv \mathsf{if}\ p\ \mathsf{then}\ (f; \mathsf{if}\ q\ \mathsf{then}\ \bot\ \mathsf{else}\ \mathsf{id})\ \mathsf{else}\ \bot,$$

where $\bot$ is the program that always diverges, and id is the program that does nothing. Thus, any Hoare assertion can be encoded as an equation. Again, we want to allow hypotheses of the form $\{p\}a\{q\}$, where a is an atomic action, and hypotheses that are propositional formulas for tests. More formally, the properties we consider are expressed by implications $\Phi \Rightarrow \{p\}f\{q\}$, where Φ is a list of hypotheses. The set of such implications that are true under any interpretation is called the *Hoare theory* of monadic recursion schemes. This Hoare theory is the primary object of study for the present paper.

At a technical level, our work is closely related to the line of work on the propositional fragment of Hoare logic, called *Propositional Hoare Logic* or PHL. This logic was introduced by Kozen in [16, 17], where it is shown to be subsumed by Kleene algebra with tests (KAT) [15], a propositional Horn equational system that combines Kleene algebra (KA) [13, 14] with Boolean algebra. Moreover, it is proved that PHL is PSPACE-complete (see also [5]) and therefore as complex to decide as the more expressive KAT, which is also PSPACE-complete. A deductive Hoare-style calculus for a variant of PHL is presented in [18], which is sound and complete for the set of relationally valid implications of the form

$$\{p_1\}a_1\{q_1\},\ \ldots,\ \{p_k\}a_k\{q_k\} \Rightarrow \{p\}f\{q\},$$

where $a_1, \ldots, a_k$ are atomic actions and f is an arbitrary regular program (built using the operations of composition ;, nondeterministic choice $+$, and nondeterministic iteration *). Contrary to the present paper, both PHL and KAT are concerned with iteration and do not handle arbitrary recursion.

We note that the result of [16, 17] on the subsumption of PHL by KAT suggests that for practical reasoning purposes KAT offers an expressiveness advantage over PHL with no complexity increase. However, if we add a recursion operator to the language of KAT, then arbitrary context-free languages can be expressed. This means that the equational theory can have no recursive axiomatization and no decision procedure. The increased complexity of such an equational theory that combines nondeterminism and recursion raises the need for identification of more computationally manageable fragments.

Related to both PHL and KAT is Propositional Dynamic Logic (PDL), which is a modal logic for reasoning about regular programs. Standard PDL only concerns programs with iteration and is already EXPTIME-complete. Extensions of PDL with recursive programs can be highly complex. For example, its extension with the context-free program $\{a^i b a^i \mid i \geq 0\}$ is Π^1_1-complete. Much more on the subject of non-regular PDL can be found in [10].

Our contribution. We investigate the simple implicational theory and the Hoare theory of monadic recursion schemes. Our results are the following:

- We show that the simple implicational theory of monadic recursion schemes is decidable and, in fact, primitive recursive. This extends the known result about the decidability of the equational theory.
- We give a sound and complete Hoare-style calculus for the Hoare theory of monadic recursion schemes. We also obtain an analogous result for monadic while schemes.
- A decision procedure is given for the Hoare theory that requires exponential time. Moreover, it is shown that the Hoare theory is EXPTIME-hard.

2. Preliminaries

Monadic recursion schemes can be given as a collection of equations, which are to be thought of as mutually recursive parameterless procedure declarations. For example,

$$X \triangleq \mathsf{if}\ p\ \mathsf{then}\ a; X; Y; b\ \mathsf{else}\ c$$
$$Y \triangleq \mathsf{if}\ q\ \mathsf{then}\ a\ \mathsf{else}\ c; Y; X; d$$

where p, q are abstract atomic tests and a, b, c, d are abstract atomic actions. The procedure symbol X is designated as the start symbol. Alternatively, such schemes can be given as terms that involve the recursion operation μ. The μ operation binds program variables, e.g. $\mu X.\mathsf{if}\ p\ \mathsf{then}\ a; X; b$ corresponds to the recursive definition

$$X \triangleq \mathsf{if}\ p\ \mathsf{then}\ a; X; b.$$

There are straightforward translations from one formalism to the other that only incur a polynomial blow-up in size. These translations are related to Bekić's theorem (see Chapter 10 of [26] for an elementary exposition).

2.1 The language of monadic program schemes

The language of monadic program schemes involves two sorts: the sort of *tests*, and the sort of *programs*. Tests are built up from *atomic tests* and the constants true and false, using the test operations $\neg$ (negation), $\wedge$ (conjunction), and $\vee$ (disjunction). We typically use the letters $p, q, \ldots$ to range over arbitrary tests. So, the tests are given by the grammar

$$p, q ::= \text{atomic test} \mid \mathsf{true} \mid \mathsf{false} \mid \neg p \mid p \wedge q \mid p \vee q.$$

As usual, the implication $p \rightarrow q$ is treated as abbreviation for $\neg p \vee q$, and the double implication $p \leftrightarrow q$ as abbreviation for $(p \rightarrow q) \wedge (q \rightarrow p)$. The base programs are *atomic programs* $a, b, \ldots$ (also called *atomic actions*), *program variables* $X, Y, \ldots$, and the constants id and $\bot$, called *skip* and *fail* respectively. Compound programs are constructed using the operations ;, if, while, and μ, called *(sequential) composition*, *conditional*, *iteration*, and *recursion* respectively. For a test p, a program variable X, and ar-

bitrary programs f, g, the following are programs:

$$f;g \qquad \mathsf{if}\, p\, \mathsf{then}\, f\, \mathsf{else}\, g \qquad \mathsf{while}\, p\, \mathsf{do}\, f \qquad \mu X.f$$

For notational brevity, we will sometimes write $p[f,g]$ instead of $\mathsf{if}\, p\, \mathsf{then}\, f\, \mathsf{else}\, g$, and $\mathsf{w}pf$ instead of $\mathsf{while}\, p\, \mathsf{do}\, f$.

2.2 Denotational semantics of programs

We present the standard denotational semantics of monadic program schemes. Every program term is interpreted as a partial function on a set representing an abstract state space. Every test is interpreted as a unary predicate on the state space.

Notation 1. Before we give the formal semantics of the language, we present some relevant notation and definitions. We assume that a partial function $f : A \rightharpoonup B$ from a set A to a set B is represented as a binary relation $\{(x, f(x)) \mid x \in A, f(x) \text{ defined}\} \subseteq A \times B$. We use the arrow $\rightharpoonup$ instead of $\to$ to indicate partiality. The domain of a partial function f, denoted $\mathsf{dom}f$, is defined as $\mathsf{dom}f = \{x \in A \mid f(x) \text{ defined}\}$. If the partial functions $f, g : A \rightharpoonup B$ have disjoint domains, then their union is a partial function $f \cup g : A \rightharpoonup B$. More generally, if $f_i : A \rightharpoonup B$ is an arbitrary family of partial functions with pairwise disjoint domains $\mathsf{dom}f_i$, then $\bigcup_i f_i : A \rightharpoonup B$ is a well-defined partial function. We write Id_A for the identity relation (total function) on A, that is, $Id_A = \{(x,x) \mid x \in A\}$. The operation of composition of partial functions is written as **;** (boldface ;). The operands are given in diagrammatic order:

$$\frac{f : A \rightharpoonup B \qquad g : B \rightharpoonup C}{f;g : A \rightharpoonup C}.$$

For partial functions $f, g : A \rightharpoonup B$, we write $f \leq g$ when for every $x \in A$ with $f(x)$ defined, it holds that $g(x)$ is also defined and is equal to $f(x)$. The partial order $\leq$ is sometimes referred to as the *extension order* on partial functions. For a partial function $f : A \rightharpoonup A$ with $f \leq Id_A$ we define its *complement* $\sim f : A \rightharpoonup A$ as $\sim f = Id_A \setminus f$. Finally, for a partial function $h : A \rightharpoonup A$, we define $h^n : A \rightharpoonup A$ to be the n-fold composite of h. That is, $h^0 = Id_A$ and $h^{n+1} = h^n; h$. A straightforward induction establishes that $h^{n+1} = h; h^n$ for every $n \geq 0$.

An *interpretation* of the language of monadic program schemes consists of a set A, called the *domain*, and an interpretation function I. Elements of the domain A are called *states*, and we use lowercase letters $x, y, \ldots$ to range over them. We think of A as being the state space of the program. For a program f, its interpretation $I(f) : A \rightharpoonup A$ is a partial function from A to A. For $x \in A$, if $I(f)(x)$ is undefined, then this is to be understood as divergence or failure of the program when started at state x. The interpretation of a test p is $I(p) : A \rightharpoonup A$. Intuitively, $I(p)(x) = x$ when $p(x)$ is true, and $I(p)(x)$ is undefined when $p(x)$ is false. Alternatively, $I(p)$ can be represented as a unary predicate on A, that is, as a subset of A. The interpretation function specifies the meaning $I(a) : A \rightharpoonup A$ and $I(p) : A \rightharpoonup A$ for every atomic program a and every atomic test p. Moreover, it specifies the meaning $I(X) : A \rightharpoonup A$ of (some of) the program variables. Now, we describe how I extends to all tests and programs. For tests:

$$\begin{aligned} I(\mathsf{true}) &= Id_A & I(\neg p) &= \sim I(p) \\ I(\mathsf{false}) &= \emptyset & I(p \wedge q) &= I(p); I(q) \\ & & I(p \vee q) &= I(p) \cup I(q) \end{aligned}$$

For programs, we define I inductively as follows:

$$\begin{aligned} I(\mathsf{id}) &= Id_A & I(f;g) &= I(f); I(g) \\ I(\bot) &= \emptyset & I(p[f,g]) &= I(p); I(f) \cup \sim I(p); I(g) \\ & & I(\mathsf{w}pf) &= \textstyle\bigcup_{n\geq 0} (I(p); I(f))^n; \sim I(p) \\ & & I(\mu X.f) &= \textstyle\bigcup_{n\geq 0} \tau_n \end{aligned}$$

where $\tau_0 = \emptyset$ and $\tau_{n+1} = I[X \mapsto \tau_n](f)$. The notation $I[X \mapsto \tau_n]$ denotes the function that agrees with I, except possibly for X which is mapped to τ_n. Equivalently, $I(\mu X.f)$ can be defined as the least fixpoint of the monotone map $\sigma \mapsto I[X \mapsto \sigma](f)$, for $\sigma : A \rightharpoonup A$.

Claim 2. $I(\mathsf{w}pf) = \bigcup_{n\geq 0} I(p[f, \mathsf{id}])^n; \sim I(p)$.

Claim 3. Let X be a program variable not appearing free in the program f. Then, $I(\mathsf{w}pf) = I(\mu X.p[f; X, \mathsf{id}])$.

Claim 3 says that every while loop can be equivalently written using recursion. So, for some of our results we do not need to take while as a primitive operator. We have chosen to include it as a primitive symbol, because we will present a complete Hoare calculus for while program schemes, that is, schemes in which we do not allow general recursion.

2.3 Formulas and their semantics

First, we consider the notions of satisfaction and validity for tests. Let I be an interpretation of the language of programs. For a test p and a state x, we write $I, x \models p$ when $I(p)(x) = x$. We read: "the state x satisfies p (under the interpretation I)". When $I, x \models p$ for every state x, we say that I satisfies p and we write $I \models p$. Finally, we say that p is valid when $I \models p$ for every interpretation I, and we write $\models p$.

An *equation* is an expression $f \equiv g$, where f and g are programs. An interpretation I satisfies the equation $f \equiv g$, denoted as $I \models f \equiv g$, if $I(f) = I(g)$. An equation $f \equiv g$ is *valid*, written $\models f \equiv g$, when every interpretation satisfies it.

A *Hoare assertion* is an expression $\{p\}f\{q\}$, where p, q are tests and f is a program. Informally, it says that when the program starts at a state satisfying the predicate p and f terminates, then the state after the execution of f satisfies the predicate q. This intuition is formalized as follows: for all states x, y in the domain, $I, x \models p$ and $I(f)(x) = y$ imply that $I, y \models q$. In this case we write $I \models \{p\}f\{q\}$ and say that I satisfies the Hoare assertion $\{p\}f\{q\}$. We typically use letters $\phi, \psi, \ldots$ to range over Hoare assertions. A Hoare assertion is called *simple* if it is of the form $\{p\}a\{q\}$ or $\{p\}X\{q\}$, where a is an atomic action and X is a program variable.

Remark 4. Directly from the definitions we get that $I \models \{p\}f\{q\}$ iff $I(p); I(f); \sim I(q) = \emptyset$. Since $I(p) = I(p[\mathsf{id}, \bot])$ and $\sim I(p) = I(p[\bot, \mathsf{id}])$ for every test p, we observe that

$$\begin{aligned} I(p); I(f); \sim I(q) &= I(p[\mathsf{id}, \bot]); I(f); I(q[\bot, \mathsf{id}]) \\ &= I(p[\mathsf{id}, \bot]; f; q[\bot, \mathsf{id}]) \\ &= I(p[f; q[\bot, \mathsf{id}], \bot]). \end{aligned}$$

So, $I \models \{p\}f\{q\}$ iff $I \models p[f; q[\bot, \mathsf{id}], \bot] \equiv \bot$. This reduces the satisfaction of a Hoare assertion to the satisfaction of an equation.

We use capital Greek letters $\Phi, \Psi, \ldots$ to denote collections of simple Hoare assertions and tests. We say that I satisfies such a collection Φ, and write $I \models \Phi$, if I satisfies every Hoare assertion and every test in Φ. We are only concerned here with implications of one of the following forms:

$$\Phi \Rightarrow f \equiv g \qquad \Phi \Rightarrow p \qquad \Phi \Rightarrow \{p\}f\{q\}$$

where Φ is a finite collection of simple Hoare assertions and tests, p is a test, and $\{p\}f\{q\}$ is an arbitrary Hoare assertion. We call them *simple implications*. Implications of the two last forms in particular are called *Hoare implications*. An interpretation I satisfies an implication $\Phi \Rightarrow \phi$, written $I \models \Phi \Rightarrow \phi$, if $I \models \Phi$ implies that $I \models \phi$. An implication $\Phi \Rightarrow \phi$ is *valid*, which we denote by $\models \Phi \Rightarrow \phi$, if every interpretation satisfies it.

The set of valid equations $f \equiv g$ between monadic recursive schemes is the *equational theory* of such schemes. We denote this set by EqTheory. The set SImpTheory of valid simple implications $\Phi \Rightarrow f \equiv g$ is the *simple implicational theory* of monadic recursive schemes. Analogously, we call the set of valid Hoare implications $\Phi \Rightarrow p$ and $\Phi \Rightarrow \{p\}f\{q\}$ the *Hoare theory* of monadic schemes. We write this set as HoareTheory.

3. The simple implicational theory

We augment equations $f \equiv g$ between schemes with hypotheses Φ that are either tests or simple Hoare assertions $\{p\}a\{q\}$. The main result of this section is that the validity of such an implication $\Phi \Rightarrow f \equiv g$ can be reduced to the validity of a simple equation $\widetilde{f} \equiv \widetilde{g}$. The reduction can incur an exponential blow-up in size. The idea of the proof is to replace each atomic action a by a program $\widetilde{a}$ that in some sense encodes the hypotheses Φ.

Suppose that Φ is a finite collection of tests and simple Hoare assertions. We assume that we only have a finite number of atomic tests $p_1, p_2, \ldots, p_k$ available. For each i, take ℓ_i to be the literal p_i or $\neg p_i$. We call the sequence $\ell_1\ell_2\cdots\ell_k$ an *atom*. We use lowercase letters $\alpha, \beta, \gamma, \ldots$ from the beginning of the Greek alphabet to range over atoms. An atom is essentially a conjunction of literals, and it can thought of as a propositional truth assignment. We write $\alpha \leq p$ or $\alpha \models p$ to mean that the atom α satisfies the test p. An atom α is *consistent with* Φ or Φ-*consistent* if $\alpha \models p$ for every test $p \in \Phi$. Let At_c be the set of atoms that are consistent with Φ.

Remark 5. Let I be an interpretation just for the tests of the language, $\Phi = \{q_1, \ldots, q_n\}$ be a collection of tests, and At_c be the corresponding set of Φ-consistent atoms. We observe that I satisfies Φ iff $I \models \bigvee At_c$. By Boolean logic, every test q_i is equivalent to a disjunction of atoms. That is, $\models q_i \leftrightarrow \alpha^i_1 \vee \cdots \vee \alpha^i_{m_i}$. Let $At^i_c = \{\alpha^i_1, \ldots, \alpha^i_{m_i}\}$ be the atoms satisfying q_i. Then, the set of Φ-consistent atoms is $At_c = \bigcap_i At^i_c$. It suffices to show that $\bigwedge_i(\alpha^i_1 \vee \cdots \vee \alpha^i_{m_i})$ is equivalent to $\bigvee At_c$. Indeed,

$$\begin{aligned}\models \textstyle\bigwedge_i(\alpha^i_1 \vee \cdots \vee \alpha^i_{m_i}) &\leftrightarrow \textstyle\bigwedge_i \left[(\bigvee At_c) \vee \bigvee(At^i_c \setminus At_c)\right] \\ &\leftrightarrow (\textstyle\bigvee At_c) \vee \bigwedge_i \bigvee(At^i_c \setminus At_c),\end{aligned}$$

which is equivalent to $\bigvee At_c$. So, the disjunct $\bigvee At_c$ is the disjunctive normal form of (and hence, equivalent to) the conjunction of tests in Φ.

Definition 6 (the $\widetilde{\cdot}$ transformation)**.** Fix an atomic action a. For a Φ-consistent atom α we define q_α to be the conjunct

$$q_\alpha = \bigwedge\{q \mid \{p\}a\{q\} \in \Phi,\ \alpha \leq p\}.$$

Intuitively, q_α is the test that has to hold (according to Φ) after executing the action a from a state satisfying the atom α. Define now the program $\widetilde{a}$ as the following case statement:

$$\begin{aligned}\widetilde{a} = \ &\mathsf{if}\ \alpha\ \mathsf{then}\ (a; \mathsf{if}\ (q_\alpha \wedge \textstyle\bigvee At_c)\ \mathsf{then\ id\ else}\ \bot) \\ &\mathsf{else\ if}\ \beta\ \mathsf{then}\ (a; \mathsf{if}\ (q_\beta \wedge \textstyle\bigvee At_c)\ \mathsf{then\ id\ else}\ \bot) \\ &\mathsf{else\ if}\ \ldots \\ &\mathsf{else\ if}\ \gamma\ \mathsf{then}\ (a; \mathsf{if}\ (q_\gamma \wedge \textstyle\bigvee At_c)\ \mathsf{then\ id\ else}\ \bot) \\ &\mathsf{else}\ \bot,\end{aligned}$$

where the atoms $\alpha, \beta, \ldots, \gamma$ in the statement are exactly the Φ-consistent atoms. We will extend the $\widetilde{\cdot}$ transformation now to arbitrary programs. First, define

$$\widetilde{\mathsf{id}} = \mathsf{if}\ (\textstyle\bigvee At_c)\ \mathsf{then\ id\ else}\ \bot.$$

Define the substitution θ to map every atomic program a to $\widetilde{a}$, and the constant id to $\widetilde{\mathsf{id}}$. For a program f, we put $\widetilde{f} = \theta(f)$.

Lemma 7. Let I be an interpretation that satisfies Φ. Then, it holds that $I(\widetilde{a}) = I(a)$ for an atomic action a. In fact, $I(\widetilde{f}) = I(f)$ for every program term f.

Let $S \subseteq A$ and $f : A \rightharpoonup A$ be a partial function. We say that f is S-*restricted* if both the domain and the range of f are contained in S. This means that we can essentially view f as a partial function of type $f : S \rightharpoonup S$.

Lemma 8. Let I be an interpretation, and S be the subset of states that satisfy $\bigvee At_c$. We assume that for every program variable X, $I(X)$ is S-restricted. Then, for every program term f, the partial function $I(\widetilde{f})$ is S-restricted.

Theorem 9. The implication $\Phi \Rightarrow f \equiv g$ is valid iff the equation $\widetilde{f} \equiv \widetilde{g}$ is valid.

Proof. For the right-to-left direction, suppose that the equation $\widetilde{f} \equiv \widetilde{g}$ is valid and consider an interpretation I that satisfies Φ. We have that $I \models \widetilde{f} \equiv \widetilde{g}$, that is, $I(\widetilde{f}) = I(\widetilde{g})$. We have to show that I satisfies $f \equiv g$. This follows immediately from Lemma 7, which says that $I(f) = I(\widetilde{f}) = I(\widetilde{g}) = I(g)$.

For the left-to-right direction, suppose that $\Phi \Rightarrow f \equiv g$ is valid and consider an arbitrary interpretation I. We have to show that $I(\widetilde{f}) = I(\widetilde{g})$. W.l.o.g. we can assume that the programs f and g have no free program variables, and that no program variable appears in Φ. Lemma 8 says that $I(\widetilde{f})$ and $I(\widetilde{g})$ remain essentially unchanged if we restrict the domain to the states that satisfy $\bigvee At_c$. So, w.l.o.g. we can assume from now on that $I(\bigvee At_c) = Id$, that is, I satisfies all the tests in Φ. This means that $I(\mathsf{id}) = I(\mathsf{id}) = Id$. Now, we modify the interpretation so that every atomic action a is mapped to the partial function $I(\widetilde{a})$. We thus define

$$I' = I[a \mapsto I(\widetilde{a}), \text{for all } a].$$

The interpretation I' now satisfies Φ by construction of $\widetilde{a}$. Since $\Phi \Rightarrow f \equiv g$ has been assumed to be valid, we get that I' satisfies $f \equiv g$. That is, $I'(f) = I'(g)$. By a straightforward "substitution lemma" we have that

$$I'(f) = I[a \mapsto I(\widetilde{a}), \text{for all } a](f) = I(\widetilde{f})$$

and similarly that $I'(g) = I(\widetilde{g})$. It follows that $I(\widetilde{f}) = I(\widetilde{g})$. So, the equation $\widetilde{f} \equiv \widetilde{g}$ is valid. □

Corollary 10. SImpTheory is decidable. In fact, it is a primitive recursive set.

Proof. Theorem 9 essentially says that the simple implicational theory SImpTheory of monadic recursion schemes can be reduced to their equational theory EqTheory. The reduction produces an equation of size exponential in the size of the implication. This is because each atomic program a is replaced by a case statement $\widetilde{a}$ whose size is proportional to the number of atoms in At_c. Decidability follows from the result of Sénizergues on the decidability of the language-equivalence problem for DPDAs [20]. The problems of DPDA equivalence and (strong) equivalence of monadic recursion schemes are interreducible, as shown in [8] and [7]. The primitive recursive upper bound follows from the result of Stirling [25], which is a strengthening of the decidability result for DPDA equivalence. □

4. A sound and complete Hoare calculus

In this section we propose a Hoare-style calculus (Table 1) which is sound and complete for the Hoare theory of monadic recursion schemes. The completeness proof is based on a standard technique in logic: we define a "free" interpretation whose theory is exactly

$$\frac{\{p\}a\{q\} \text{ in } \Phi}{\Phi \vdash \{p\}a\{q\}} \qquad \frac{}{\Phi \vdash \{p\}\mathsf{id}\{p\}} \qquad \frac{}{\Phi \vdash \{\mathsf{true}\}\bot\{\mathsf{false}\}}$$

$$\frac{\Phi \vdash \{p\}f\{q\} \qquad \Phi \vdash \{q\}g\{r\}}{\Phi \vdash \{p\}f;g\{r\}}$$

$$\frac{\Phi \vdash \{p \wedge q\}f\{r\} \qquad \Phi \vdash \{p \wedge \neg q\}g\{r\}}{\Phi \vdash \{p\}\mathsf{if}\ q\ \mathsf{then}\ f\ \mathsf{else}\ g\{r\}}$$

$$\frac{\Phi \vdash \{p \wedge q\}f\{p\}}{\Phi \vdash \{p\}\mathsf{while}\ q\ \mathsf{do}\ f\{p \wedge \neg q\}}$$

$$\frac{\Phi[\mathrm{X}\colon \{p_j\}X\{q_j\} \text{ for } j \in J] \vdash \{p_k\}f\{q_k\} \text{ for every } k \in J}{\Phi \vdash \{p_\ell\}\mu X.f\{q_\ell\}}$$

$$\frac{\Phi \vdash p' \to p \qquad \Phi \vdash \{p\}f\{q\} \qquad \Phi \vdash q \to q'}{\Phi \vdash \{p'\}f\{q'\}}$$

$$\frac{\Phi \vdash \{p\}f\{q_1\} \qquad \Phi \vdash \{p\}f\{q_2\}}{\Phi \vdash \{p\}f\{q_1 \wedge q_2\}} \qquad \frac{}{\Phi \vdash \{p\}f\{\mathsf{true}\}}$$

$$\frac{\Phi \vdash \{p_1\}f\{q\} \qquad \Phi \vdash \{p_2\}f\{q\}}{\Phi \vdash \{p_1 \vee p_2\}f\{q\}} \qquad \frac{}{\Phi \vdash \{\mathsf{false}\}f\{q\}}$$

Table 1. Proof system for deriving Hoare implications.

the set of valid Hoare implications. It is called "free" because it is free of extra properties: if an implication is satisfied in it, then it is satisfied in every interpretation.

We define a proof system in Table 1 with which we derive Hoare implications. We use the notation $\Phi \vdash \phi$ to mean that the Hoare implication $\Phi \Rightarrow \phi$ is provable in our system. In the premise of the μ-rule in Table 1 appears the notation

$$\Phi[X : \{p_j\}X\{q_j\} \text{ for } j \in J],$$

which denotes the set that results from Φ by replacing any Hoare assertions for X by the assertions $\{p_j\}X\{q_j\}$ for $j \in J$. The index set J is always assumed to be finite. We assume that we have a complete calculus for Boolean logic. So, $\Phi \models p$ implies that $\Phi \Rightarrow p$ is provable, where p is a test.

Proposition 11. The Hoare-style calculus of Table 1 is sound.

Proof. Verifying that the rules in Table 1 are sound is completely standard, except possibly for the μ-rule. So, we will only give the proof for the soundness of the μ-rule. Let Ψ denote the collection of Hoare assertions

$$\Phi[X : \{p_j\}X\{q_j\} \text{ for } j \in J].$$

Suppose that $\Psi \models \{p_k\}f\{q_k\}$ for every $k \in J$. We show that

$$\Phi \models \{p_\ell\}\mu X.f\{q_\ell\}$$

for an arbitrary $\ell \in J$. Let I be an interpretation for which $I \models \Phi$. We have to show that $I \models \{p_\ell\}\mu X.f\{q_\ell\}$.

Recall that $I(\mu X.f) = \bigcup_{n\geq 0} \tau_n$, where $\tau_0 = \emptyset$ and $\tau_{n+1} = I[X \mapsto \tau_n](f)$. We claim that

$$I(p_k); \tau_n; {\sim}I(q_k) = \emptyset \text{ for every } k \in J \text{ and } n \geq 0.$$

The proof is by induction on n. For the base case, it clearly holds that $I(p_k); \tau_0; {\sim}I(q_k) = \emptyset$ because $\tau_0 = \emptyset$. For the induction step assume that $I(p_k); \tau_n; {\sim}I(q_k) = \emptyset$ for every $k \in J$. Since $I \models \Phi$, I satisfies all the Hoare assertions in Ψ that do not involve X. Now, notice that for every $k \in J$:

$$\begin{aligned} &I(p_k); \tau_n; {\sim}I(q_k) = \emptyset \iff \\ &I(p_k); I[X \mapsto \tau_n](X); {\sim}I(q_k) = \emptyset \iff \\ &I[X \mapsto \tau_n] \models \{p_k\}X\{q_k\}. \end{aligned}$$

It follows that $I[X \mapsto \tau_n] \models \Psi$, and therefore $I[X \mapsto \tau_n] \models \{p_k\}f\{q_k\}$ for every $k \in J$. So, for every $k \in J$:

$$\begin{aligned} &I[X \mapsto \tau_n] \models \{p_k\}f\{q_k\} \iff \\ &I(p_k); I[X \mapsto \tau_n](f); I(q_k) = \emptyset \iff \\ &I(p_k); \tau_{n+1}; {\sim}I(q_k) = \emptyset. \end{aligned}$$

Using the claim we have just proved we can show that

$$\begin{aligned} I(p_\ell); I(\mu X.f); I(q_\ell) &= I(p_\ell); \left(\bigcup_{n\geq 0} \tau_n\right); I(q_\ell) \\ &= \bigcup_{n\geq 0} I(p_\ell); \tau_n; I(q_\ell) = \emptyset. \end{aligned}$$

It follows that $I \models \{p_\ell\}\mu X.f\{q_\ell\}$, and we are done. □

Let us give some intuition for the crucial rule for recursion. It can be thought of as corresponding to a proof by induction, where the claim is multi-part. For the induction step, we argue under the hypotheses Φ augmented with the induction hypothesis: for every j in the finite set J, it holds that $\{p_j\}X\{q_j\}$. We show that every part of the claim is preserved:

$$\Phi[X : \{p_j\}X\{q_j\} \text{ for } j \in J] \vdash \{p_k\}f\{q_k\},$$

for every $k \in J$. We thus conclude that every part of the claim is satisfied by the recursive procedure $\mu X.f$ (under the hypotheses Φ). That is, $\Phi \vdash \{p_\ell\}\mu X.f\{q_\ell\}$ for every $\ell \in J$.

4.1 Free interpretation

Consider a finite collection Φ of tests and simple Hoare assertions. We will see how to construct an interpretation I_Φ, which depends on Φ, that satisfies the following properties:

(1) I_Φ satisfies every test and assertion in Φ.

(2) For a test q, if $I_\Phi \models q$ then $\Phi \vdash q$.

(3) For a Hoare assertion ϕ, if $I_\Phi \models \phi$ then $\Phi \vdash \phi$.

The existence of such an interpretation, which we call a *free interpretation*, has as an easy consequence the completeness of the calculus given in Table 1, as we will see later.

Remark 12. For showing the existence of a free interpretation, we can assume without loss of generality that the Hoare assertions in Φ are of the form $\{\alpha\}a\{q\}$ or $\{\alpha\}X\{q\}$, where α is a Φ-consistent atom.

Definition 13 (free interpretation)**.** Fix a finite collection Φ of tests and simple Hoare assertions, that is, assertions of the form $\{p\}a\{q\}$ or $\{p\}X\{q\}$. We define the domain A_Φ of the *free interpretation* for Φ to be the set At_c^+ of all finite non-empty strings over the set At_c of Φ-consistent atoms. Intuitively, a state $\alpha_1\alpha_2\ldots\alpha_n$ gives us the atom currently satisfied (α_1), as well as the atoms that will be true after each execution of an atomic action. When the string is a single atom, the computation is expected to terminate. Since the first atom of a state is meant to indicate the currently satisfied atom, we interpret an atomic test as follows:

$$I_\Phi(p)(\alpha x) = \begin{cases} \alpha x, & \text{if } \alpha \leq p; \\ \text{undefined}, & \text{if } \alpha \leq \neg p. \end{cases}$$

We need to consider now the interpretation of the atomic actions and of the program variables. The Hoare assumptions in Φ restrict

the atoms that are *reachable* via an action a or X. For Φ-consistent atoms α, β and for an atomic program a, define:

$$\alpha \xrightarrow{a}_\Phi \beta \quad \text{iff} \quad \beta \leq q, \text{for all } \{\alpha\}a\{q\} \in \Phi.$$

So, $\alpha \xrightarrow{a}_\Phi \beta$ means that β can be reached from α via a under the restriction that the assumptions Φ are satisfied. So, for an atomic program a we define:

$$I_\Phi(a)(\alpha\beta x) = \begin{cases} \beta x, & \text{if } \alpha \xrightarrow{a}_\Phi \beta \\ \text{undefined}, & \text{otherwise} \end{cases}$$

and $I_\Phi(a)(\alpha)$ is undefined (because a single-atom state α signifies that the computation should have terminated). For a program variable X, we define $\alpha \xrightarrow{X}_\Phi \beta$ and $I_\Phi(X)$ analogously.

Lemma 14. The free interpretation I_Φ satisfies Φ.

4.2 Atoms reachable via programs

Already in the definition of the interpretation of an atomic action in I_Φ we introduced the notion of an atom β being *reachable* from an atom α via the program a, and we denoted this by $\alpha \xrightarrow{a}_\Phi \beta$. This is meant to correspond to the idea that a state satisfying α can be transformed to a state satisfying β when the action a is executed. We will extend the notion of reachable atoms to arbitrary programs. For every program f, we will define inductively a function $\mathsf{ps}_\Phi(f) : At_c \to \wp At_c$, where $\wp$ denotes the powerset operation. The map $\mathsf{ps}_\Phi(f)$ sends an atom α to the set $\mathsf{ps}_\Phi(f)(\alpha)$ of atoms that are *reachable via* f.

Notation 15. For functions of type $At_c \to \wp At_c$, we define a binary sum operation $+$ and a corresponding arbitrary sum operation $\sum$ as follows:

$$\frac{\sigma : At_c \to \wp At_c \qquad \tau : At_c \to \wp At_c}{\sigma + \tau := \lambda\alpha \in At_c.\ \sigma(\alpha) \cup \tau(\alpha) : At_c \to \wp At_c}$$

$$\frac{\sigma_j : At_c \to \wp At_c \qquad j \in J}{\sum_j \sigma_j := \lambda\alpha \in At_c.\ \bigcup_{j\in J} \sigma_j(\alpha) : At_c \to \wp At_c}$$

We also consider a composition operation $;$ given by

$$\frac{\sigma : At_c \to \wp At_c \qquad \tau : At_c \to \wp At_c}{\sigma;\tau := \lambda\alpha \in At_c.\ \bigcup_{\beta\in\sigma(\alpha)} \tau(\beta) : At_c \to \wp At_c}.$$

The operation $;$ is associative with left and right unit the function $\alpha \mapsto \{\alpha\}$. The operation $+$ is associative, commutative, and idempotent. The unit for $+$ is the function $\alpha \mapsto \emptyset$. We have left and right distributivity of $;$ over $+$:

$$\sigma;(\tau_1 + \tau_2) = \sigma;\tau_1 + \sigma;\tau_2$$
$$(\sigma_1 + \sigma_2);\tau = \sigma_1;\tau + \sigma_2;\tau$$

In fact, stronger distributivity properties hold: $\sigma; \sum_j \tau_j = \sum_j \sigma;\tau_j$ and $(\sum_j \sigma_j);\tau = \sum_j \sigma_j;\tau$. For a function $\sigma : At_c \to \wp At_c$ we define σ^n to be the n-fold composite of σ. That is, $\sigma^0 = \mathsf{ps}(\mathsf{id})$ and $\sigma^{n+1} = \sigma^n;\sigma$. It holds that $\sigma^{n+1} = \sigma;\sigma^n$ for every $n \geq 0$. We define a partial order on functions of type $At_c \to \wp At_c$ by: $\sigma \leq \tau$ iff $\sigma + \tau = \tau$. Observe that $\sigma \leq \tau$ iff $\sigma(\alpha) \subseteq \tau(\alpha)$ for every $\alpha \in At_c$.

Notation 16. Let Φ be a finite collection of tests and simple Hoare assertions (of the form $\{p\}a\{q\}$ or $\{p\}X\{q\}$). Fix a program variable X and a function $\sigma : At_c \to \wp At_c$. We denote by $\Phi[X : \sigma]$ the set that results from Φ by removing all assertions involving X and replacing them by assertions that agree with σ. That is, we put the assertions $\{\alpha\}X\{\bigvee \sigma(\alpha)\}$ for every $\alpha \in At_c$. From Definition 13 we see that $\alpha \xrightarrow{X}_{\Phi[X:\sigma]} \beta$ iff $\beta \in \sigma(\alpha)$ for every $\alpha, \beta \in At_c$. So, the free interpretation $I_{\Phi[X:\sigma]}$ is equal to the modified free interpretation $I_\Phi[X : \tilde{\sigma}]$, where $\tilde{\sigma} : At_c^+ \rightharpoonup At_c^+$ is defined as follows:

$$\tilde{\sigma}(\alpha\beta x) = \begin{cases} \beta x, & \text{if } \beta \in \sigma(\alpha) \\ \text{undefined}, & \text{otherwise} \end{cases}$$

and $\tilde{\sigma}(X)(\alpha)$ is undefined for every $\alpha \in At_c$. Also, notice that $\mathsf{ps}_{\Phi[X:\sigma]}(X) = \sigma$.

Definition 17. For an arbitrary test q, we define the function $\mathsf{ps}_\Phi(q) : At_c \to \wp At_c$ by $\mathsf{ps}_\Phi(q)(\alpha) = \{\alpha\}$ when $\alpha \leq q$, and $\mathsf{ps}_\Phi(q)(\alpha) = \emptyset$ when $\alpha \leq \neg q$. Now, we define $\mathsf{ps}_\Phi(f) : At_c \to \wp At_c$ by induction on the structure of the program:

$$\mathsf{ps}_\Phi(\mathsf{id})(\alpha) = \{\alpha\} \qquad \mathsf{ps}_\Phi(a)(\alpha) = \{\beta \mid \alpha \xrightarrow{a}_\Phi \beta\}$$
$$\mathsf{ps}_\Phi(\bot)(\alpha) = \emptyset \qquad \mathsf{ps}_\Phi(X)(\alpha) = \{\beta \mid \alpha \xrightarrow{X}_\Phi \beta\}$$
$$\mathsf{ps}_\Phi(f;g) = \mathsf{ps}_\Phi(f);\mathsf{ps}_\Phi(g)$$
$$\mathsf{ps}_\Phi(p[f,g]) = \mathsf{ps}_\Phi(p);\mathsf{ps}_\Phi(f) + \mathsf{ps}_\Phi(\neg p);\mathsf{ps}_\Phi(g)$$
$$\mathsf{ps}_\Phi(\mathsf{w}pf) = \textstyle\sum_{n\geq 0}(\mathsf{ps}_\Phi(p);\mathsf{ps}_\Phi(f))^n;\mathsf{ps}_\Phi(\neg p)$$
$$\mathsf{ps}_\Phi(\mu X.f) = \textstyle\sum_{n\geq 0} \sigma_n$$

where $\sigma_0 = \lambda\alpha.\emptyset$ and $\sigma_{n+1} = \mathsf{ps}_{\Phi[X:\sigma_n]}(f)$. To extend the notation we have been using for atomic actions to arbitrary programs, we write $\alpha \xrightarrow{f}_\Phi \beta$ when $\mathsf{ps}_\Phi(f)(\alpha) = \beta$.

Observation 18. Immediately from unfolding the definition of $\mathsf{ps}_\Phi(f;g)$ and $\mathsf{ps}_\Phi(p[f,g])$ we obtain that:

$$\mathsf{ps}_\Phi(f;g)(\alpha) = \textstyle\bigcup_{\beta\in\mathsf{ps}_\Phi(f)(\alpha)} \mathsf{ps}_\Phi(g)(\beta)$$
$$\mathsf{ps}_\Phi(p[f,g])(\alpha) = \begin{cases} \mathsf{ps}_\Phi(f)(\alpha), & \text{if } \alpha \leq p \\ \mathsf{ps}_\Phi(g)(\alpha), & \text{if } \alpha \leq \neg p \end{cases}$$

The first of the above equations says that $\alpha \xrightarrow{f}_\Phi \beta \xrightarrow{g}_\Phi \gamma$ implies $\alpha \xrightarrow{f;g}_\Phi \gamma$. The second equation says that if $\alpha \leq p$ and $\alpha \xrightarrow{f}_\Phi \beta$, then $\alpha \xrightarrow{p[f,g]}_\Phi \beta$. Similarly, if $\alpha \leq \neg p$ and $\alpha \xrightarrow{g}_\Phi \beta$, then we have $\alpha \xrightarrow{p[f,g]}_\Phi \beta$.

Claim 19. The following equations hold:

$$\mathsf{ps}(p[f,\mathsf{id}])^n;\mathsf{ps}(\neg p) = \textstyle\sum_{0\leq i\leq n}(\mathsf{ps}(p);\mathsf{ps}(f))^i;\mathsf{ps}(\neg p)$$
$$\mathsf{ps}(\mathsf{w}pf) = \textstyle\sum_{n\geq 0} \mathsf{ps}(p[f,\mathsf{id}])^n;\mathsf{ps}(\neg p)$$

We have dropped the Φ subscripts to reduce notational clutter.

Claim 20. Let X be a program variable not appearing free in the program f. Then, $\mathsf{ps}_\Phi(\mathsf{w}pf) = \mathsf{ps}_\Phi(\mu X.p[f;X,\mathsf{id}])$.

Lemma 21 (monotonicity)**.** Let $\sigma_i, \tau_i : At_c \to \wp At_c$ be a finite collection of pairs of functions. If $\sigma_i \leq \tau_i$ for every i, then $\mathsf{ps}_{\Phi[X_i:\sigma_i]}(f) \leq \mathsf{ps}_{\Phi[X_i:\tau_i]}(f)$.

A consequence of the monotonicity property above is that the approximants $\{\sigma_n \mid n \geq 0\}$ for $\mathsf{ps}_\Phi(\mu X.f) = \bigcup_{n\geq 0} \sigma_n$ form a countable chain $\sigma_0 \leq \sigma_1 \leq \sigma_2 \leq \cdots$. The claim is that $\sigma_n \leq \sigma_{n+1}$ for every $n \geq 0$. The base case $\sigma_0 = \lambda\alpha.\emptyset \leq \sigma_1$ is obvious. For the induction step we need to show that $\sigma_{n+1} \leq \sigma_{n+2}$, which is equivalent to the inequality $\mathsf{ps}_{\Phi[X:\sigma_n]}(f) \leq \mathsf{ps}_{\Phi[X:\sigma_{n+1}]}(f)$. But this is a consequence of the induction hypothesis $\sigma_n \leq \sigma_{n+1}$ and of Lemma 21.

Lemma 22 (continuity)**.** Let $\sigma_0 \leq \sigma_1 \leq \cdots$ be a countably infinite chain of functions $At_c \to \wp At_c$, and $\tau = \sum_{n\geq 0} \sigma_n$. Then, $\mathsf{ps}_{\Phi[X:\tau]}(f) = \sum_{n\geq 0} \mathsf{ps}_{\Phi[X:\sigma_n]}(f)$.

Proposition 23. For any Φ-consistent atom α and any program f, it holds that $\Phi \vdash \{\alpha\}f\{\bigvee \mathsf{ps}_\Phi(f)(\alpha)\}$.

Proof. We note that the Hoare assertion is well-formed, because the set of atoms $\mathsf{ps}_\Phi(f)(\alpha)$ is finite. The proof proceeds by induction on the structure of the program. For the skip program we have that $\Phi \vdash \{\alpha\}\mathsf{id}\{\alpha\}$ (axiom of the system) and $\bigvee \mathsf{ps}_\Phi(\mathsf{id})(\alpha) = \bigvee\{\alpha\} = \alpha$. For the fail program, we have the derivation

$$\frac{\Phi \vdash \alpha \to \mathsf{true} \qquad \Phi \vdash \{\mathsf{true}\}\bot\{\mathsf{false}\}}{\Phi \vdash \{\alpha\}\bot\{\mathsf{false}\}}$$

and $\bigvee \mathsf{ps}_\Phi(\bot)(\alpha) = \bigvee \emptyset = \mathsf{false}$. For an atomic program a, we recall that $\mathsf{ps}_\Phi(a)(\alpha) = \{\beta \mid \alpha \xrightarrow{a} \beta\}$, which is equal to

$$\{\beta \in At_c \mid \beta \leq q \text{ for all } \{\alpha\}a\{q\} \in \Phi\}.$$

Notice that $\Phi \models \bigvee \mathsf{ps}_\Phi(a)(\alpha) \leftrightarrow \bigwedge_{\{\alpha\}a\{q\}\in\Phi} q$. Using the and-rule, we get the derivation

$$\frac{\Phi \vdash \{\alpha\}a\{q\}, \text{ where } \{\alpha\}a\{q\} \in \Phi}{\Phi \vdash \{\alpha\}a\{\bigwedge_{\{\alpha\}a\{q\}\in\Phi} q\}},$$

and the weakening rule gives us $\Phi \vdash \{\alpha\}a\{\bigvee \mathsf{ps}_\Phi(a)(\alpha)\}$. We have thus covered all the base cases.

We consider now the case of the composite $f;g$. The induction hypothesis gives us that $\Phi \vdash \{\alpha\}f\{\bigvee \mathsf{ps}_\Phi(f)(\alpha)\}$. For every atom β in $\mathsf{ps}_\Phi(f)(\alpha)$, we have from the I.H. that $\Phi \vdash \{\beta\}g\{\bigvee \mathsf{ps}_\Phi(g)(\beta)\}$, and therefore

$$\Phi \vdash \{\beta\}g\{\bigvee_{\beta\in\mathsf{ps}_\Phi(f)(\alpha)} \bigvee \mathsf{ps}_\Phi(g)(\beta)\}$$

by the weakening rule. We observe that

$$\begin{aligned}\models \bigvee_{\beta\in\mathsf{ps}_\Phi(f)(\alpha)} \bigvee \mathsf{ps}_\Phi(g)(\beta) &\leftrightarrow \bigvee \bigcup_{\beta\in\mathsf{ps}_\Phi(f)(\alpha)} \mathsf{ps}_\Phi(g)(\beta)\\ &\leftrightarrow \bigvee \mathsf{ps}_\Phi(f;g)(\alpha),\end{aligned}$$

and consequently the or-rule gives us that

$$\Phi \vdash \{\bigvee \mathsf{ps}_\Phi(f)(\alpha)\}g\{\bigvee \mathsf{ps}_\Phi(f;g)(\alpha)\}.$$

Now, we use the composition rule, and we obtain that $\Phi \vdash \{\alpha\}f;g\{\bigvee \mathsf{ps}_\Phi(f;g)(\alpha)\}$.

We handle now the case of the conditional $p[f,g]$. Suppose that $\alpha \leq p$. We want to show that

$$\Phi \vdash \{\alpha\}p[f,g]\{\bigvee \mathsf{ps}_\Phi(p[f,g])(\alpha)\}.$$

Since $\alpha \leq p$, we have that $\mathsf{ps}_\Phi(p[f,g])(\alpha) = \mathsf{ps}_\Phi(f)(\alpha)$. Using the induction hypothesis and the fact that $\models \alpha \wedge p \leftrightarrow \alpha$ we have the derivation

$$\frac{\Phi \vdash \alpha \wedge p \to \alpha \qquad \Phi \vdash \{\alpha\}f\{\bigvee \mathsf{ps}_\Phi(f)(\alpha)\}}{\Phi \vdash \{\alpha \wedge p\}f\{\bigvee \mathsf{ps}_\Phi(f)(\alpha)\}}.$$

Moreover, we have that $\models \alpha \wedge \neg p \leftrightarrow \mathsf{false}$, which gives us the derivation

$$\frac{\Phi \vdash \alpha \wedge \neg p \to \mathsf{false} \qquad \Phi \vdash \{\mathsf{false}\}g\{\bigvee \mathsf{ps}_\Phi(f)(\alpha)\}}{\Phi \vdash \{\alpha \wedge \neg p\}g\{\bigvee \mathsf{ps}_\Phi(f)(\alpha)\}}.$$

Now, using the rule for conditionals we conclude that $\Phi \vdash \{\alpha\}p[f,g]\{\bigvee \mathsf{ps}_\Phi(f)(\alpha)\}$. The case of $\alpha \leq \neg p$ is handled similarly and we omit it.

We handle the case of a loop $\mathsf{w}pf$. Intuitively, the idea is to consider the set of all atoms that appear during the execution of the loop as the loop invariant. From Claim 19 and the distributivity property we see that

$$\mathsf{ps}_\Phi(\mathsf{w}pf) = \underbrace{\left(\textstyle\sum_{n\geq 0} \mathsf{ps}_\Phi(p[f,\mathsf{id}])^n\right)}_{\tau : At_c \to \wp At_c} ; \mathsf{ps}_\Phi(\neg p).$$

Let $\alpha \in At_c$. We take the disjunction of the set of atoms $\tau(\alpha)$ to be the loop invariant. So, we want to show that $\Phi \vdash \{r \wedge p\}f\{r\}$, where $r = \bigvee \tau(\alpha)$. Consider an atom $\beta \in \tau(\alpha)$ with $\beta \leq p$. There exists $n \geq 0$ with $\beta \in \mathsf{ps}(p[f,\mathsf{id}])^n(\alpha)$. By the I.H., $\Phi \vdash \{\beta\}f\{\bigvee \mathsf{ps}(f)(\beta)\}$. Now, we claim that $\mathsf{ps}(f)(\beta) \subseteq \tau(\alpha)$ and therefore the implication $\bigvee \mathsf{ps}(f)(\beta) \to r$ is valid. Since $\beta \leq p$, we have that $\mathsf{ps}(f)(\beta) = \mathsf{ps}(p[f,\mathsf{id}])(\beta)$. So,

$$\begin{aligned}\mathsf{ps}(f)(\beta) &= \mathsf{ps}(p[f,\mathsf{id}])(\beta)\\ &\subseteq \textstyle\bigcup_{\beta'\in\mathsf{ps}(p[f,\mathsf{id}])^n(\alpha)} \mathsf{ps}(p[f,\mathsf{id}])(\beta')\\ &= (\mathsf{ps}(p[f,\mathsf{id}])^n ; \mathsf{ps}(p[f,\mathsf{id}]))(\alpha)\\ &= \mathsf{ps}(p[f,\mathsf{id}])^{n+1}(\alpha),\end{aligned}$$

which is contained in $\tau(\alpha)$. The weakening rule gives us that $\Phi \vdash \{\beta\}f\{r\}$. Since we have considered any Φ-consistent atom $\beta \leq r \wedge p$, by the or-rule, we get that

$$\Phi \vdash \{\textstyle\bigvee_{\beta\leq r\wedge p,\ \beta\in At_c} \beta\}f\{r\}.$$

From $\Phi \models r \wedge p \leftrightarrow \bigvee\{\beta \in At_c \mid \beta \leq r \wedge p\}$ and the weakening rule, we have $\Phi \vdash \{r \wedge p\}f\{r\}$. By the iteration rule,

$$\Phi \vdash \{r\}\mathsf{w}pf\{r \wedge \neg p\}.$$

From the expression that is given in Claim 19 for $\mathsf{ps}(\mathsf{w}pf)$ we obtain the validities

$$\models \bigvee \mathsf{ps}(\mathsf{w}pf)(\alpha) \leftrightarrow (\bigvee \tau(\alpha)) \wedge \neg p \leftrightarrow r \wedge \neg p,$$

and hence $\Phi \vdash \{r\}\mathsf{w}pf\{\bigvee \mathsf{ps}(\mathsf{w}pf)(\alpha)\}$. Finally, from $\alpha \in \tau(\alpha)$ we have that $\alpha \to r$ is valid. The weakening rule then gives us $\Phi \vdash \{\alpha\}\mathsf{w}pf\{\bigvee \mathsf{ps}(\mathsf{w}pf)(\alpha)\}$.

It remains to consider the case $\mu X.f$ of recursion. Let γ be an arbitrary Φ-consistent atom, and $\tau = \mathsf{ps}_\Phi(\mu X.f)$. We want to show that $\Phi \vdash \{\gamma\}\mu X.f\{\bigvee \tau(\gamma)\}$. By the μ-rule it suffices to prove that $\Psi \vdash \{\beta\}f\{\bigvee \tau(\beta)\}$ for every $\beta \in At_c$, where

$$\Psi = \Phi[X:\tau] = \Phi[X : \{\alpha\}X\{\bigvee \tau(\alpha)\} \text{ for } \alpha \in At_c].$$

Recall that $\tau = \bigcup_{n\geq 0} \sigma_n$, where $\sigma_0 = \lambda\alpha.\emptyset$ and $\sigma_{n+1} = \mathsf{ps}_{\Phi[X:\sigma_n]}(f)$. From Lemma 22 we obtain that

$$\begin{aligned}\mathsf{ps}_\Psi(f) = \mathsf{ps}_{\Phi[X:\tau]}(f) &= \textstyle\sum_{n\geq 0} \mathsf{ps}_{\Phi[X:\sigma_n]}(f)\\ &= \textstyle\sum_{n\geq 0} \sigma_{n+1} = \tau.\end{aligned}$$

So, the induction hypothesis gives us that $\Psi \vdash \{\beta\}f\{\bigvee \tau(\beta)\}$, and the proof is complete. □

4.3 Strongest postconditions & completeness

Let p be a test, f be a program, and I be an interpretation. We define $\mathsf{post}_I(p,f)$ to be the set of states that can be reached via f from a state satisfying p. That is,

$$\mathsf{post}_I(p,f) = \{I(f)(x) \mid I(f)(x) \text{ defined}, I,x \models p\}.$$

It is straightforward to show the equivalence: $I \models \{p\}f\{q\}$ iff $\mathsf{post}_I(p,f) \subseteq \{y \mid I,y \models q\}$. For the particular case where we consider a free interpretation I_Φ and p is a Φ-consistent atom α, we have that

$$\mathsf{post}_\Phi(\alpha,f) = \{I_\Phi(f)(\alpha x) \mid x \in At^*\},$$

where we have omitted for the sake of brevity the condition that $I_\Phi(f)(\alpha x)$ has to be defined. For notational convenience, we write post_Φ instead of post_{I_Φ}. Recall that the set $\mathsf{ps}_\Phi(f)(\alpha)$ contains the Φ-consistent atoms that are reachable from α via f. The set $\mathsf{post}_\Phi(\alpha,f)$ contains the states reachable via f from a state satisfying α. These sets are related in a useful way. The exact relationship between them is given by Proposition 26 and Claim 25.

Definition 24. Let Φ be a collection of tests and simple Hoare assertions, and I be an interpretation with domain $A_\Phi = At_c^+$. We say that I *is consistent with* Φ if the following hold:

(1) For every atomic test p and every atom $\alpha \in At_c$, it holds that $I(p)(\alpha x) = \alpha x$ iff $\alpha \leq p$.

(2) For every atomic program a and every atom $\alpha \in At_c$, the equation $\mathsf{post}_I(\alpha, a) = \mathsf{ps}_\Phi(a)(\alpha) \cdot At_c^*$ holds.
(3) For every program variable X and every atom $\alpha \in At_c$, the equation $\mathsf{post}_I(\alpha, X) = \mathsf{ps}_\Phi(X)(\alpha) \cdot At_c^*$ holds.

Claim 25. Let Φ be a collection of tests and simple Hoare assertions. The free interpretation I_Φ is consistent with Φ.

Proposition 26. Let Φ be a finite collection of tests and simple Hoare assertions, and I be an interpretation that is consistent with Φ. For every program f and every atom $\alpha \in At_c$ we have that $\mathsf{post}_I(\alpha, f) = \mathsf{ps}_\Phi(f)(\alpha) \cdot At_c^*$.

Theorem 27 (completeness)**.** Let Φ be a finite collection of tests and simple Hoare assertions, and $\{p\}f\{q\}$ be a Hoare assertion. The following are equivalent:

(1) $\Phi \models \{p\}f\{q\}$.
(2) $I_\Phi \models \{p\}f\{q\}$.
(3) $\mathsf{ps}_\Phi(f)(\alpha) \subseteq \{\beta \in At_c \mid \beta \leq q\}$ for every atom $\alpha \in At_c$ with $\alpha \leq p$.
(4) $\Phi \vdash \{p\}f\{q\}$.

Proof. We need only consider the atomic tests $p_1, p_2, \ldots, p_k$ that appear in Φ and $\{p\}f\{q\}$. So, only finitely many atomic tests are relevant.

We show (1) $\Rightarrow$ (2). Let I_Φ be the free interpretation for Φ. Since I_Φ satisfies Φ (Lemma 14) and $\Phi \models \{p\}f\{q\}$ (by our hypothesis), we have that $I_\Phi \models \{p\}f\{q\}$.

We show (2) $\Rightarrow$ (3). Using Proposition 26 and Claim 25, it is easy to see that for every atom $\alpha \in At_c$ with $\alpha \leq p$, $I_\Phi \models \{\alpha\}f\{q\}$ is equivalent to:

$$\mathsf{post}_\Phi(\alpha, f) \subseteq \{\beta y \in At_c^+ \mid I_\Phi, \beta y \models q\} \iff$$
$$\mathsf{ps}_\Phi(f)(\alpha) \cdot At_c^* \subseteq \{\beta \in At_c \mid \beta \leq q\} \cdot At_c^* \iff$$
$$\mathsf{ps}_\Phi(f)(\alpha) \subseteq \{\beta \in At_c \mid \beta \leq q\}.$$

The implication (2) $\Rightarrow$ (3) follows immediately.

We show (3) $\Rightarrow$ (4). By construction of I_Φ, we know that $I_\Phi \models q \leftrightarrow \bigvee\{\beta \in At_c \mid \beta \leq q\}$. So, the inclusion

$$\mathsf{ps}_\Phi(f)(\alpha) \subseteq \{\beta \in At_c \mid \beta \leq q\}$$

is equivalent to $I_\Phi \models \bigvee \mathsf{ps}(f)(\alpha) \rightarrow q$. By the assumed completeness of the Boolean calculus included in $\vdash$, this in turn is equivalent to $\Phi \vdash \bigvee \mathsf{ps}(f)(\alpha) \rightarrow q$. Intuitively, we have seen by now that the function $\mathsf{ps}_\Phi(f)$ gives us strongest postconditions for the program f in the free interpretation. Suppose now that $I_\Phi \models \{p\}f\{q\}$. Let α be a Φ-consistent atom with $\alpha \leq p$. We have that $I_\Phi \models \{\alpha\}f\{q\}$. From the previous discussion, we have that $\Phi \vdash \bigvee \mathsf{ps}(f)(\alpha) \rightarrow q$. From Proposition 23 we have that $\Phi \vdash \{\alpha\}f\{\bigvee \mathsf{ps}(f)(\alpha)\}$. Using the weakening axiom, we get $\Phi \vdash \{\alpha\}f\{q\}$. Now, we make use of the or-rule:

$$\frac{\Phi \vdash \{\alpha\}f\{q\}, \text{for all } \alpha \in At_c \text{ with } \alpha \leq p}{\Phi \vdash \{\bigvee\{\alpha \in At_c \mid \alpha \leq p\}\}f\{q\}}.$$

But $I_\Phi \models p \leftrightarrow \bigvee\{\alpha \in At_c \mid \alpha \leq p\}$, and we thus conclude that $\Phi \vdash \{p\}f\{q\}$.

Finally, the implication (4) $\Rightarrow$ (1) is the soundness of our Hoare calculus (Proposition 11). □

The implications (2) $\Rightarrow$ (3) $\Rightarrow$ (4) shown in Theorem 27 and Lemma 14 (together with the assumed complete Boolean calculus included in $\vdash$) say that the free interpretation I_Φ indeed satisfies the three desirable properties that we stated in the beginning of Section 4.1. We easily obtain completeness of the suggested Hoare-style calculus by showing the implication (1) $\Rightarrow$ (2) in Theorem 27.

Theorem 28 (completeness for while schemes)**.** Consider the restriction of the language of monadic schemes to while programs, that is, recursion μ and program variables do not appear. The Hoare calculus of Table 1 (with the μ-rule removed) is complete for the Hoare theory of these schemes.

Proof. We simply observe that everything goes through with essentially no change when arbitrary recursion is removed and iteration is retained. □

We note that Theorem 28 is related to but does not follow from the completeness result of Kozen and Tiuryn [18] for the Propositional Hoare Logic of regular programs. First, we consider here a different language with no non-determinism. Additionally, our semantics is deterministic, whereas in [18] interpretations are allowed to be arbitrary relations. Showing completeness for a smaller class of interpretations is, of course, a stronger result.

5. Complexity of the Hoare theory

We investigate the computational complexity of the problem μHOARE: "Given a finite set Φ of tests and simple Hoare assertions and a Hoare assertion $\{p\}f\{q\}$, is it the case that $\Phi \models \{p\}f\{q\}$?"

Theorem 29. The problem μHOARE is in EXPTIME.

Proof. Since $\Phi \models \{p\}f\{q\}$ is equivalent to $\Phi \models \{\alpha\}f\{q\}$ for every atom $\alpha \in At_c$ with $\alpha \leq p$, we can restrict attention to Hoare consequences of the form $\{\alpha\}f\{q\}$.

As we showed in Theorem 27, the statement $\Phi \models \{\alpha\}f\{q\}$, where α is an atom in At_c, is equivalent to the containment

$$\mathsf{ps}_\Phi(f)(\alpha) \subseteq \{\beta \in At_c \mid \beta \leq q\}.$$

Recall that $\mathsf{ps}_\Phi(f)(\alpha)$ is the set of atoms that are reachable from α via f (Definition 17). So, the problem amounts to computing the function $\mathsf{ps}_\Phi(f) : At_c \rightarrow \wp At_c$ for arbitrary programs f (where $\wp$ denotes the powerset operation). We will give a procedure for computing an explicit representation of $\mathsf{ps}_\Phi(f)$.

Let k be the number of atomic tests that appear in the input $\Phi, \{p\}f\{q\}$. The size N of the set At_c of Φ-consistent atoms is bounded above by 2^k (the number of all atoms). Given an arbitrary atom α, we can decide in linear time whether α is Φ-consistent, because we simply check if α satisfies all the tests in Φ. Notice that we can represent a function $At_c \rightarrow \wp At_c$ as a $At_c \times At_c$ matrix with entries 0 or 1. With this representation the operation **;** corresponds to matrix multiplication. Such a multiplication takes time $O(N^3)$. We define the "if" operation for a test p and matrices σ, τ as:

$$p[\sigma, \tau]_{\alpha\beta} = \begin{cases} \sigma_{\alpha\beta}, & \text{if } \alpha \leq p \\ \tau_{\alpha\beta}, & \text{if } \alpha \leq \neg p \end{cases}$$

Computing $p[\sigma, \tau]$ takes time $O(N^2)$.

We give a recursive algorithm $PS(g, \{\sigma_a\}_a, \{\sigma_X\}_X)$ that takes as input a program g, a finite collection $\{\sigma_a\}_a$ of matrices for all the atomic programs, and a finite collection $\{\sigma_X\}_X$ of matrices for all the program variables. We use $\bar{\sigma}$ as an abbreviation for $\{\sigma_a\}_a, \{\sigma_X\}_X$. For most of the cases, the algorithm can be described with simple equations:

$$PS(b, \bar{\sigma}) \triangleq \sigma_b \qquad PS(\mathsf{id}, \bar{\sigma}) \triangleq \mathbf{1}_{At_c}$$
$$PS(Y, \bar{\sigma}) \triangleq \sigma_Y \qquad PS(\bot, \bar{\sigma}) \triangleq \mathbf{0}_{At_c}$$
$$PS(g; h, \bar{\sigma}) \triangleq PS(g, \bar{\sigma}); PS(h, \bar{\sigma})$$
$$PS(p[g, h], \bar{\sigma}) \triangleq p[PS(g, \bar{\sigma}), PS(h, \bar{\sigma})]$$

where $\mathbf{0}_{At_c}$ is the matrix with 0's everywhere, and $\mathbf{1}_{At_c}$ has 1's on the diagonal and 0's elsewhere. We describe the case $\mu Y.g$ in a

$$
\begin{array}{l}
PS(\mu Y.g, \bar{\sigma}) \triangleq \{ \\
\quad S := \mathbf{0}_{At_c} \\
\quad \mathsf{for}\ t = 1, \ldots, N \cdot N\ \mathsf{do} \\
\qquad newS := PS(g, \bar{\sigma}[Y \mapsto S]) \\
\qquad S := newS \\
\quad \mathsf{return}\ S \\
\}
\end{array}
$$

Figure 1. Definition of the recursive procedure $PS(f, \bar{\sigma})$ for the case $f = \mu Y.g$.

more operational way in Figure 1. By $\bar{\sigma}[Y \mapsto S]$ we denote the modification of $\bar{\sigma}$ that maps Y to the matrix S. It is straightforward to see that

$$PS(f, \bar{\sigma}) = \mathsf{ps}_\Phi(f),$$

where $\sigma_a = \mathsf{ps}_\Phi(a)$ and $\sigma_X = \mathsf{ps}_\Phi(X)$. In the case of recursion we just need to observe that a matrix has $N \cdot N$ entries and therefore the fixpoint $\mathsf{ps}_\Phi(\mu Y.g)$ is reached within $N \cdot N$ iterations.

We calculate an exponential upper bound for the running time of the recursive algorithm. We think of the tree of recursive calls. Let n be the size of the program g. At every recursive call the size of the program reduces strictly. So, the depth of the tree is bounded above by n. As far as branching of the tree is concerned, the worst case is when we have recursion. The branching in that case is $N \cdot N$. The time needed to combine the results of the recursive subtrees is $O(N^3)$ (worst case when we have multiplication). So, we have an asymptotic upper bound $N^3 \cdot (N \cdot N)^n$ for the running time of the algorithm. Since $N \leq 2^k$, we have a less tight bound $(2^k)^3 \cdot (2^k \cdot 2^k)^n = 2^{3k+2kn}$, which is exponential in the size of the input. Calculating the initial values for $\bar{\sigma}$ from Φ can clearly be done in exponential time. □

Theorem 30. The problem μHOARE is EXPTIME-hard.

Proof. We show how to encode the computations of polynomial-space bounded alternating Turing machines [4]. Consider a machine with states $Q = Q_{\text{and}} \cup Q_{\text{or}}$ (partitioned into and-states & or-states), input alphabet Σ, tape alphabet Γ, blank symbol $\_$, start state q_0, and transition relation

$$\Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{-1, 0, +1\}).$$

A transition $\langle (q, a), (q', b, d) \rangle \in \Delta$ says that if the machine is in state q and is scanning the symbol a, then it spawns a new process with its own copy of the tape in which the state is set to q', the symbol b is written over the current position, and the cursor moves by d. If $d = -1$ ($d = +1$) the cursor moves one position to the left (right), and if $d = 0$ the cursor stays in the same position. The machine accepts (rejects) if it halts at an and-state (or-state).

The idea is to simulate the alternating machine with a recursive program, where recursive calls correspond to the existential and universal branching of the machine. After every recursive call the tape is restored to exactly what it was before the call. In this way we simulate parallel branching in which each process has its own copy of the tape. Without loss of generality we can assume that every computation path halts.

We introduce atomic tests P_i^a for every tape symbol a and every position i. Intuitively, P_i^a is true when the tape has symbol a at position i. The tests

$$\textstyle\bigwedge_i \bigvee_a P_i^a \ \text{ and } \ \bigwedge_i \bigwedge_{a \neq b} \neg(P_i^a \wedge P_j^b)$$

say that every position is associated with a unique symbol. The atomic test A is used for returning the result of each recursive call. The test A is true iff the machine accepts. We introduce program variables $X[q, i]$ for every state q and every position i. We think of $X[q, i]$ as the procedure corresponding to the machine being in state q and at position i. Similarly, we introduce the variables $Y[q, i, a]$, where q, i have the same interpretation as before and a corresponds to the currently scanned symbol. So, we define $X[q, i]$ as a case statement that invokes the appropriate $Y[q, i, a]$:

$$
\begin{array}{rl}
X[q,i] \triangleq & \mathsf{if}\ P_i^a\ \mathsf{then}\ Y[q,i,a] \\
& \mathsf{else\ if}\ P_i^b\ \mathsf{then}\ Y[q,i,b] \\
& \mathsf{else}\ \ldots
\end{array}
$$

We introduce atomic programs accept and reject that set the test A to true and false respectively. Moreover, they leave all other tests unchanged. So, we take the assumptions

$$
\begin{array}{ll}
\{\mathsf{true}\}\mathsf{accept}\{A\} & \{\mathsf{true}\}\mathsf{reject}\{\neg A\} \\
\{P_i^a\}\mathsf{accept}\{P_i^a\} & \{P_i^a\}\mathsf{reject}\{P_i^a\}
\end{array}
$$

where i ranges over all positions and a over all tape symbols. Now, if (q, a) has no Δ-successor and q is an and-state we define $Y[q, i, a] \triangleq \mathsf{accept}$. Similarly, if (q, a) has no Δ-successor and q is an or-state we define $Y[q, i, a] \triangleq \mathsf{reject}$. Suppose that q is an and-state and that (q, a) has exactly two Δ-successors:

$$(q, a)\Delta(q_1, b_1, d_1) \quad \text{and} \quad (q, a)\Delta(q_2, b_2, d_2).$$

We define the procedure $Y[q, i, a]$ as shown in Figure 2. If q is an or-state with (q, a) having exactly two Δ-successors (q_1, b_1, d_1) and (q_2, b_2, d_2), the procedure $Y[q, i, a]$ is defined analogously (see Figure 3). The generalization to more than two Δ-successors is straightforward. The atomic program 'write b at i' writes the symbol b at the position i of the tape and leaves everything else unchanged. We can express this with the following assumptions:

$$
\begin{array}{ll}
\{\mathsf{true}\}\mathsf{write}\ b\ \mathsf{at}\ i\{P_i^b\} & \\
\{P_i^a\}\mathsf{write}\ b\ \mathsf{at}\ i\{P_i^a\} & (\text{for all } a \neq b) \\
\{A\}\mathsf{write}\ b\ \mathsf{at}\ i\{A\} & \\
\{\neg A\}\mathsf{write}\ b\ \mathsf{at}\ i\{\neg A\} &
\end{array}
$$

For input string $x_1 x_2 \cdots x_n$ we define the test $start$, which encodes the initial tape, as

$$start = P_1^{x_1} \wedge \cdots \wedge P_n^{x_n} \wedge \overline{P}_{n+1} \wedge \cdots \wedge \overline{P}_{\pi(n)},$$

where $\pi(n)$ is the polynomial that gives the space bound of the machine. We have given a collection of mutually recursive functions $X[q, i]$ and $Y[q, i, a]$. This can be turned into a program term using the μ-operator in the standard way. Since the space is bounded by a polynomial $\pi(n)$, there are polynomially many positions i. So, the size of the program is polynomial in the size of the machine. Finally, the claim is that the machine accepts iff $\Phi \models \{start\}X[q_0, 1]\{A\}$, where Φ is the collection of our assumptions for the atomic tests and the atomic programs. □

$$
\begin{array}{ll}
Y[q,i,a] \triangleq & \mathsf{write}\ b_1\ \mathsf{at}\ i; \quad \text{// write } b_1 \text{ at current position} \\
& X[q_1, i + d_1]; \quad \text{// spawn first child process} \\
& \mathsf{write}\ a\ \mathsf{at}\ i; \quad \text{// restore tape} \\
& \mathsf{if}\ A\ \mathsf{then}\ \{ \quad \text{// first process accepted} \\
& \quad \mathsf{write}\ b_2\ \mathsf{at}\ i; \quad \text{// write } b_2 \text{ at current position} \\
& \quad X[q_2, i + d_2]; \quad \text{// spawn second child process} \\
& \quad \mathsf{write}\ a\ \mathsf{at}\ i; \quad \text{// restore tape} \\
& \quad \text{// result} = A \\
& \}\ \mathsf{else}\ \{ \quad \text{// first process rejected} \\
& \quad \mathsf{id} \quad \text{// propagate failure upwards} \\
& \}
\end{array}
$$

Figure 2. Encoding universal branching with recursive calls.

```
Y[q, i, a] ≜ write b_1 at i;          // write b_1 at current position
             X[q_1, i + d_1];         // spawn first child process
             write a at i;            // restore tape
             if (¬A) then {           // first process rejected
                 write b_2 at i;      // write b_2 at current position
                 X[q_2, i + d_2];     // spawn second child process
                 write a at i;        // restore tape
                 // result = A
             } else {                 // first process accepted
                 id                   // propagate success upwards
             }
```

Figure 3. Encoding existential branching with recursive calls.

6. Conclusion

We have shown that the simple implicational theory of monadic recursion schemes reduces to their equational theory, with an exponential blow-up. We also considered the propositional Hoare theory of such schemes, and we obtained a sound and complete calculus. Finally, the Hoare theory was shown to be EXPTIME-complete.

References

[1] S. Böhm and S. Göller. Language equivalence of deterministic real-time one-counter automata is NL-complete. In *Mathematical Foundations of Computer Science (MFCS 2011)*, pages 194–205. 2011.

[2] S. Böhm, S. Göller, and P. Jančar. Equivalence of deterministic one-counter automata is NL-complete. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*, pages 131–140, 2013.

[3] S. Böhm, S. Göller, and P. Jancar. Equivalence of deterministic one-counter automata is NL-complete. *CoRR*, abs/1301.2181, 2013. URL `http://arxiv.org/abs/1301.2181`.

[4] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

[5] E. Cohen and D. Kozen. A note on the complexity of propositional Hoare logic. *ACM Transactions on Computational Logic*, 1(1):171–174, 2000.

[6] M. Davis. *Computability and Unsolvability*. Dover Publications, 1982.

[7] E. P. Friedman. Equivalence problems for deterministic context-free languages and monadic recursion schemes. *Journal of Computer and System Sciences*, 14(3):344–359, 1977.

[8] S. J. Garland and D. C. Luckham. Program schemes, recursion schemes, and formal languages. *Journal of Computer and System Sciences*, 7(2):119–160, 1973.

[9] S. Ginsburg and S. Greibach. Deterministic context free languages. *Information and Control*, 9(6):620–648, 1966.

[10] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[11] P. Jančar. A short decidability proof for DPDA language equivalence via first-order grammars. *CoRR*, abs/1010.4760, 2011. URL `http://arxiv.org/abs/1010.4760`.

[12] P. Jančar. Decidability of DPDA language equivalence via first-order grammars. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS 2012)*, pages 415–424, 2012.

[13] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. In *Proceedings of Sixth Annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 214–225, 1991.

[14] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.

[15] D. Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.

[16] D. Kozen. On Hoare logic and Kleene algebra with tests. In *Proceedings of the 14th Symposium on Logic in Computer Science (LICS 1999)*, pages 167–172, 1999.

[17] D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic*, 1(1):60–76, 2000.

[18] D. Kozen and J. Tiuryn. On the completeness of propositional Hoare logic. *Information Sciences*, 139(3-4):187–195, 2001.

[19] E. L. Post. Recursive unsolvability of a problem of Thue. *The Journal of Symbolic Logic*, 12(1):1–11, 1947.

[20] G. Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP '97)*, pages 671–681. Springer, 1997.

[21] G. Sénizergues. L(A) = L(B)? Decidability results from complete formal systems. *Theoretical Computer Science*, 251(12):1–166, 2001.

[22] G. Sénizergues. L(A) = L(B)? A simplified decidability proof. *Theoretical Computer Science*, 281(1):555–608, 2002.

[23] G. Sénizergues. The equivalence problem for t-turn DPDA is Co-NP. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, pages 478–489, 2003.

[24] C. Stirling. Decidability of DPDA equivalence. *Theoretical Computer Science*, 255(1-2):1–31, 2001.

[25] C. Stirling. Deciding DPDA equivalence is primitive recursive. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP 2002)*, pages 821–832. Springer, 2002.

[26] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, Cambridge, MA, USA, 1993.