The Hoare Logic of Deterministic and Nondeterministic Monadic Recursion Schemes*

Konstantinos Mamouras
Cornell University
`mamouras@cs.cornell.edu`

December 24, 2014

Abstract

The equational theory of deterministic 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. For this subclass of program properties, we can also handle nondeterminism at the syntactic and/or at the semantic level, without increasing the complexity of the theories. 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.

1 Introduction

The starting point of the present work is the observation that reasoning about recursive computational processes is hard. Even in the abstract propositional setting, there are fundamental barriers. Consider, for example, the familiar pushdown automata (PDAs) and the equivalent model of context-free grammars (CFGs). The equivalence problem for PDAs is a standard undecidable problem (see page 197 of [40]). In fact, PDA equivalence is Π^0_1-complete and therefore not even recursively enumerable. This implies, in particular, that there can be no effective formal system that axiomatizes the equational theory of CFGs (or of equivalent representations involving a μ least fixpoint operation). In [17] an infinitary (and hence not effective) complete axiomatization of the equational theory of context-free languages is presented.

*This is a revised and expanded version of [32].

For a special subclass of PDAs, called deterministic PDAs or DPDAs, the question of the decidability of language-equivalence was posed in [16]. After remaining open for three decades, this question was settled positively by Sénizergues in [36] (journal version [37]). Simplified proofs of this decidability result were presented later in [41] and [38]. Stirling has also obtained a primitive recursive upper bound for the problem [42], 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 [22, 21]. Jančar's proof relies on the use of first-order terms and grammars.

DPDA equivalence is related to the problem of equivalence of deterministic monadic recursion schemes (MRSs). 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 deterministic interpretation of the atomic actions and predicates. It was shown in [15] (page 132, Theorem 2.10, part (b)) that the equivalence of deterministic MRSs can be reduced to the equivalence problem for deterministic context-free grammars (these grammars correspond to DPDAs). Moreover, Friedman showed in [14] the converse, namely, that DPDA equivalence can be reduced to deterministic MRS equivalence. So, by the results of Sénizergues and Stirling, the *equational theory* of deterministic MRSs (the set of equations between such schemes that hold under every deterministic interpretation) is decidable and, in fact, it is a primitive recursive set.

The decidability of the equational theory of deterministic MRSs 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

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

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.

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

in addition to just equations. If the hypotheses are allowed to be arbitrary equations, the theory is rendered undecidable [34] (see also [10]). 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 finite collection of thus restricted hypotheses, is called the *simple implicational theory* of MRSs.

The best known algorithm for deciding DPDA equivalence, due to Stirling [42], 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 [39] 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 [5, 6]. An earlier result for deterministic real-time one-counter automata was obtained in [4].

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 $\{p\}f\{q\}$. These formulas were introduced in the seminal work of Hoare [20], which is partially based on the intermediate assertion method of Floyd [13] (see also the surveys [2, 3]). The assertion $\{p\}f\{q\}$ 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 MRSs. This Hoare theory is the primary object of study for the present paper.

As we will see, by restricting attention to program properties that can be encoded as Hoare assertions, we can allow nondeterminism at the syntactic and/or at the semantic level without increasing the computational complexity of the theories.

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 [27, 28], where it is shown to be subsumed by KAT. Kleene algebra with tests (or KAT) [25] is a propositional Horn equational system that combines Kleene algebra (KA) [23, 24] with Boolean algebra. It has been proved that PHL is PSPACE-complete (see also [8] for an alternative proof). So, PHL is 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 [31], 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.

Motivated by verification applications, several variants of KA and KAT have been considered that investigate types [26, 29] (e.g., typed KA with products), extra mutable state [18] (KAT + B!), extra equations for the primitive letters [30] (KA + Equations, or KAT + Equations), as well as the modeling of network policies for software-defined networks (SDNs) [1] (NetKAT).

We note that the result of [27, 28] 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 the system called Propositional Dynamic Logic (PDL) [11, 12], 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 [19].

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 MRSs can be reduced to their equational theory with an exponential blow-up. It follows that the simple implicational theory of deterministic MRSs is decidable and, in fact, primitive recursive. This extends the known result about the decidability of the equational theory of deterministic MRSs.

• We give a sound and complete Hoare-style calculus for the Hoare theory of MRSs. Completeness is first established w.r.t. the class of all nondeterministic interpretations, and then it is strengthened to the case of deterministic interpretations. We also obtain analogous completeness results for monadic while program schemes as a special case. All our completeness results are *unconditional* (no extra assumptions). They are not relative completeness theorems in the sense of Cook [9].

• 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,

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

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.

	while MPS (without μ)	MRS (with μ)
deterministic (without $+$)	without $+$ or μ	without $+$
nondeterministic (with $+$)	without μ	full syntax

Figure 1: Monadic Recursion Schemes: syntactic restrictions.

$\mu X.\mathsf{if}\, p\, \mathsf{then}\, a; X; b$ 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 [43] for an elementary exposition).

2.1 The language of nondeterministic monadic recursion schemes

The language of monadic recursion schemes is algebraic, and it involves two sorts: the sort of *tests*, and the sort of *programs*.

The 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

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

As usual, the implication $p \to q$ is treated as abbreviation for $\neg p \vee q$, and the double implication $p \leftrightarrow q$ as abbreviation for $(p \to q) \wedge (q \to p)$.

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 *diverge* respectively. Compound programs are constructed using the operations ;, if, while, μ, and $+$, called *(sequential) composition*, *conditional*, *iteration*, *recursion*, and *(nondeterministic) choice* respectively. The programs are thus given by the following grammar:

$$\begin{aligned}\text{programs } f,\, g ::= {} & \text{actions } a,\, b,\, \ldots \mid \text{variables } X,\, Y,\, \ldots \mid \mathsf{id} \mid \bot \mid \\ & f; g \mid \mathsf{if}\, p\, \mathsf{then}\, f\, \mathsf{else}\, g \mid \mathsf{while}\, p\, \mathsf{do}\, f \mid \mu X.f \mid f + g.\end{aligned}$$

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 $\mathsf{while}\, p\, \mathsf{do}\, f$.

In Figure 1 we summarize some syntactic restrictions of the language of nondeterministic MRSs that will be of interest for the present work. In particular, we consider the removal of the nondeterministic choice $+$ operation and/or the removal of the recursion μ operation.

2.2 Denotational semantics of programs

We present the standard denotational semantics of monadic program schemes. A nonempty set S represents the abstract state space. Every test is interpreted as a unary predicate on the state space. Every program term is interpreted as a function from S to the powerset of S. This semantics is often referred to as the *relational semantics* of programs (see [35]).

Before we give the formal semantics of the language, we present some relevant notation and definitions.

Notation 1 (nondeterministic functions)**.** For a set A, we write $\wp A$ for the *powerset* of A. A function $f : A \to \wp B$ is said to be a *nondeterministic function* from A to B. We write $f : A \rightsquigarrow B$ to denote that f is of type $A \to \wp B$. The notation $f : x \mapsto y$ means that $y \in f(x)$. For nondeterministic functions, we define a *binary sum* operation $+$, and an *arbitrary sum* operation $\sum$ as:

$$\frac{f : A \rightsquigarrow B \qquad g : A \rightsquigarrow B}{f + g \triangleq \lambda x \in A.\ f(x) \cup g(x) : A \rightsquigarrow B} \qquad \frac{f_k : A \rightsquigarrow B \qquad k \in K}{\sum_{k\in K} f_k \triangleq \lambda x \in A.\ \bigcup_{k\in K} f_k(x) : A \rightsquigarrow B}$$

The *identity function* $Id_A : A \rightsquigarrow A$ and the *zero function* $0_{AB} : A \rightsquigarrow B$ are given by:

$$Id_A \triangleq \lambda x \in A.\ \{x\} \qquad\qquad 0_{AB} \triangleq \lambda x \in A.\ \emptyset$$

We consider a *(Kleisli) composition operation* $\mathbf{;}$ with the typing rule and definition:

$$\frac{f : A \rightsquigarrow B \qquad g : B \rightsquigarrow C}{f\mathbf{;}g \triangleq \lambda x \in A.\ \bigcup_{y\in f(x)} g(y) : A \rightsquigarrow C}$$

The operation $+$ is associative, commutative, and idempotent. The unit for $+$ is the zero function 0. The operation $\mathbf{;}$ is associative with left and right unit the identity function Id. The following left and right distributivity laws hold:

$$f\mathbf{;}(g_1 + g_2) = f\mathbf{;}g_1 + f\mathbf{;}g_2 \qquad f\mathbf{;}\textstyle\sum_{k\in K} g_k = \sum_{k\in K} f\mathbf{;}g_k$$
$$(f_1 + f_2)\mathbf{;}g = f_1\mathbf{;}g + f_2\mathbf{;}g \qquad (\textstyle\sum_{k\in K} f_k)\mathbf{;}g = \sum_{k\in K} f_k\mathbf{;}g$$

For a function $f : A \rightsquigarrow A$, we define the function $f^n : A \rightsquigarrow A$ to be the *n-fold composite* of f. That is, $f^0 = Id_A$ and $f^{n+1} = f^n\mathbf{;}f$ for every $n \geq 0$. We define the *partial order* $\leq$ on functions of type $A \rightsquigarrow B$ by: $f \leq g$ iff $f + g = g$. Observe that $f \leq g$ iff ($f(x) \subseteq g(x)$ for every x in A).

Notation 2 (partial functions)**.** We say that $f : A \rightsquigarrow B$ is a *partial function*, and we write $f : A \rightharpoonup B$, if $f(x)$ is either empty or a singleton set for every $x \in A$. So, we think of $A \rightharpoonup B$ as being a subtype of $A \rightsquigarrow B$.

$$\frac{f : A \rightharpoonup B}{f : A \rightsquigarrow B}$$

We say that $f : A \rightharpoonup B$ is *defined on* $x \in A$ if $f(x) = \{y\}$ for some $y \in B$. The *domain* of the partial function $f : A \rightharpoonup B$ is given by $\mathsf{dom}\, f := \{x \in A \mid f(x) \text{ is defined}\}$.

Representing partial functions as nondeterministic functions offers economy of language, since we can use the operations we defined previously. We observe that the identity and zero functions are partial, and that nondeterministic composition $\mathbf{;}$ is composition of partial functions when applied to partial functions.

$$Id_A : A \rightharpoonup A \qquad 0_{AB} : A \rightharpoonup B \qquad \frac{f : A \rightharpoonup B \qquad g : B \rightharpoonup C}{f\mathbf{;}g : A \rightharpoonup C}$$

As operations on partial functions, the sums $+$ and $\sum$ are partial. For $f, g : A \rightharpoonup B$, the sum $f + g$ is defined when $f(x) = g(x)$ for every x in $\mathsf{dom}\, f \cap \mathsf{dom}\, g$. We say that f and g are *compatible* when their sum $f + g$ is defined. If $\mathsf{dom}\, f$ and $\mathsf{dom}\, g$ are disjoint, then f and g are clearly compatible. The order $\leq$ on $A \rightharpoonup B$ is called the *extension order.* We have that $f \leq g : A \rightharpoonup B$ when $\mathsf{dom}\, f \subseteq \mathsf{dom}\, g$ and g agrees with f on $\mathsf{dom}\, f$.

For a partial function $p : A \rightharpoonup A$ with $p \leq Id_A$ we define its *complement* $\sim\!p : A \rightharpoonup A$ to be the unique partial function with $p + \sim\!p = Id_A$. Equivalently, we have:

$$\frac{p : A \rightharpoonup A \qquad p \leq Id_A}{\sim\!p : A \rightharpoonup A} \qquad\qquad (\sim\!p)(x) \triangleq \begin{cases} \{x\}, & \text{if } p(x) = \emptyset \\ \emptyset, & \text{if } p(x) = \{x\} \end{cases}$$

Notice that a partial function $p : A \rightharpoonup A$ with $p \leq Id_A$ carries the same information as a unary predicate on A.

Definition 3 (nondeterministic interpretation)**.** An interpretation of the language of monadic program schemes consists of a set S, called the *state space*, and an *interpretation function* I. The elements of the set S are called *states*, and we use lowercase letters $x, y, \ldots$ to range over them. For a program term f, its *interpretation* $I(f) : S \rightsquigarrow S$ is a nondeterministic function on the state space.

The interpretation of a test p is a partial function $I(p) : S \rightharpoonup S$. Intuitively, $I(p)(x) = \{x\}$ when $p(x)$ is true, and $I(p)(x) = \emptyset$ when $p(x)$ is false. The interpretation function specifies the meaning of every atomic test and extends to all tests as:

$$I(\mathsf{true}) \triangleq Id_S \qquad I(\neg p) \triangleq \sim\!I(p) \qquad I(p \wedge q) \triangleq I(p); I(q)$$
$$I(\mathsf{false}) \triangleq 0_{SS} \qquad\qquad\qquad I(p \vee q) \triangleq I(p) + I(q)$$

The interpretation function specifies the meaning $I(a) : S \rightsquigarrow S$ of every atomic program a, as well as the meaning $I(X) : S \rightsquigarrow S$ of every program variable X. Now, we describe how I extends to all program terms:

$$I(\mathsf{id}) \triangleq Id_S \qquad I(f;g) \triangleq I(f); I(g) \qquad I(f+g) \triangleq I(f) + I(g)$$
$$I(\bot) \triangleq 0_{SS} \qquad I(p[f,g]) \triangleq I(p); I(f) + \sim\!I(p); I(g)$$
$$I(\mathsf{w}pf) \triangleq \textstyle\sum_{n\geq 0} \sigma_n, \text{ where} \qquad I(\mu X.f) \triangleq \textstyle\sum_{n\geq 0} \tau_n, \text{ where}$$
$$\sigma_0 \triangleq 0_{SS} \qquad \tau_0 \triangleq 0_{SS}$$
$$\sigma_{n+1} \triangleq I(p); I(f); \sigma_n + \sim\!I(p) \qquad \tau_{n+1} \triangleq I[X \mapsto \tau_n](f)$$

The expression $I[X \mapsto \tau_n]$ above denotes the interpretation that agrees with I, except possibly for X which is mapped to τ_n. We expand the definition of $I(p[f,g])$ as follows:

$$I(p[f,g])(x) = \begin{cases} I(f)(x), & \text{if } I(p)(x) = \{x\} \\ I(g)(x), & \text{if } I(p)(x) = \emptyset \end{cases}$$

for every state x in S. For nondeterministic interpretations I and I', we write $I \leq I'$ when the following hold: 1. $I(p) = I'(p)$ for every atomic test p, 2. $I(a) = I'(a)$ for every atomic action a, and 3. $I(X) \leq I'(X)$ for every program variable X.

It is easy to see in the definition of $I(\mathsf{w}pf)$ above that the while loop approximants $(\sigma_n)_{n\geq 0}$ form an increasing chain, that is $\sigma_0 \leq \sigma_1 \leq \cdots$.

Observation 4 (while loops as least fixpoints)**.** Let f be a program term, p be a test, and I be a nondeterministic interpretation with state space S. Then, the function $I(\mathsf{w}pf) : S \rightsquigarrow S$ is the least fixpoint of the map $\phi : (S \rightsquigarrow S) \to (S \rightsquigarrow S)$, which is defined by $W \mapsto I(p); I(f); W + \sim\!I(p)$.

Claim 5 (monotonicity)**.** Let I, I' be nondeterministic interpretations with $I \leq I'$ (see Definition 3 for $\leq$ on interpretations). Then, $I(f) \leq I'(f)$ for every program f.

A consequence of the monotonicity property above (Claim 5) is that the approximants $\{\tau_n \mid n \geq 0\}$ for $I(\mu X.f) = \sum_{n \geq 0} \tau_n$ form a countable chain $\tau_0 \leq \tau_1 \leq \tau_2 \leq \cdots$. The claim is that $\tau_n \leq \tau_{n+1}$ for every $n \geq 0$. The base case $\tau_0 = 0_{SS} \leq \tau_1$ is obvious. For the induction step we need to show that $\tau_{n+1} \leq \tau_{n+2}$, which is equivalent to the inequality $I[X \mapsto \tau_n](f) \leq I[X \mapsto \tau_{n+1}](f)$. But this is a consequence of the induction hypothesis $\tau_n \leq \tau_{n+1}$ and of Claim 5.

Claim 6 (continuity)**.** Let I be a nondeterministic interpretation with state space S, and $\sigma_0 \leq \sigma_1 \leq \sigma_2 \leq \cdots$ be a countable chain of functions in $S \rightsquigarrow S$. Moreover, we put $\sigma = \sum_{n \geq 0} \sigma_n$. Then, we have that $I[X \mapsto \sigma](f) = \sum_{n \geq 0} I[X \mapsto \sigma_n](f)$ for every program variable X and every program term f.

Observation 7 (recursion as least fixpoint)**.** Let f be a program term, X be a program variable, and I be a nondeterministic interpretation with state space S. Then, the function $I(\mu X.f) : S \rightsquigarrow S$ is the least fixpoint of the map $\psi : (S \rightsquigarrow S) \to (S \rightsquigarrow S)$, which is defined by $W \mapsto I[X \mapsto W](f)$.

Claim 8 (while loops & recursion)**.** Let X be a program variable not appearing free in the program f. Then, $I(\mathsf{w} p f) = I(\mu X.p[f; X, \mathsf{id}])$.

The claim above (Claim 8) says that every while loop can be written equivalently 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 operation, because we will present a complete Hoare calculus for while program schemes, that is, schemes in which we do not allow general recursion.

Definition 9 (deterministic interpretation)**.** As in the case of nondeterministic interpretations (Definition 3), a *deterministic interpretation* consists of a state space S and a function J. The interpretation of tests is exactly the same as in Definition 3. The deterministic interpretation function J specifies the meaning $J(a) : S \rightharpoonup S$ of every atomic action a, and the meaning $J(X) : S \rightharpoonup S$ of every program variable X. It extends to all program terms as in Definition 3.

We note that even though the atomic actions and the variables are interpreted by a deterministic interpretation J as partial functions, the choice operation $+$ can bring in nondeterminism. However, for $+$-free programs f, the interpretation $J(f) : S \rightharpoonup S$ is a partial function. This is shown by observing that the operations of composition, conditional, while loop, and recursion all preserve determinism.

3 Hoare Formulas and their Meaning

In this section we present formulas that are used for specifying programs, and we define their semantics. The basic formulas are the familiar *Hoare assertions* [20], which are expressions of the form $\{p\}f\{q\}$. Such formulas are also called *partial-correctness* assertions. We also consider assertions under certain hypotheses of a simple form. These formulas are called *simple Hoare implications*. The hypotheses are used for restricting the meaning of the primitive letters with axioms.

Definition 10 (tests and entailment)**.** Let I be an interpretation of tests. For a test p and a state $x \in S$, we write $I, x \models p$ when $I(p)(x) = \{x\}$. We read this as: "the state x satisfies p (under I)". When $I, x \models p$ for every state $x \in S$, we say that I *satisfies* p, and we write $I \models p$. For a set Φ of tests, the interpretation I *satisfies* Φ if it satisfies every test in Φ. We then write $I \models \Phi$. Finally, we say that Φ *entails* p, denoted $\Phi \models p$, if $I \models \Phi$ implies $I \models p$ for every I.

Definition 11 (Boolean Atoms & Φ-consistency)**.** Suppose that we have fixed a finite set of atomic tests. For an atomic test p, the expressions p and $\neg p$ are called *literals* for p (*positive* and *negative* respectively). Fix an enumeration $p_1, p_2, \ldots, p_k$ of the atomic tests. A *Boolean atom* (or simply *atom*) is an expression $\ell_1 \ell_2 \cdots \ell_k$, where every ℓ_i is a literal for p_i. 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 also be thought of as a propositional truth assignment. We write $\alpha \leq p$ to mean that the atom α satisfies the test p. We denote by At the set of all atoms.

Assume that Φ is a finite set of tests. We say that an atom α is Φ-*consistent* if $\alpha \leq p$ for every test p in Φ. We write At_Φ for the set of all Φ-consistent atoms.

Definition 12 (the free test interpretation)**.** Let Φ be a finite set of tests. We define the interpretation I_Φ on tests, which is called the *free test interpretation* w.r.t. Φ. The state space is the set At_Φ of Φ-consistent atoms, and every test represents a unary predicate on At_Φ. For an atomic test p, we define

$$I_\Phi(p)(\alpha) \triangleq \begin{cases} \{\alpha\}, & \text{if } \alpha \leq p \\ \emptyset, & \text{if } \alpha \leq \neg p \end{cases}$$

for every α in At_Φ. So, $I_\Phi(p)$ represents the set of Φ-consistent atoms that satisfy p.

An easy induction on the structure of tests proves that for every test p, the function $I_\Phi(p) : S \rightharpoonup S$ represents the set of Φ-consistent atoms that satisfy p.

Note 13 (complete Boolean calculus)**.** We assume that we have a complete Boolean calculus, with which we derive judgments $\Phi \vdash p$, where Φ is a finite set of tests and p is a test. This means that the statements

$$\Phi \models p \qquad I_\Phi \models p \qquad I_\Phi(p) = \mathsf{At}_\Phi \qquad \Phi \vdash p$$

are all equivalent. Thus, we also obtain the equivalence of the following statements:

$$\Phi \models p \to q \qquad I_\Phi \models p \to q \qquad I_\Phi(p) \leq I_\Phi(q) \qquad \Phi \vdash p \to q$$

In the proof systems that we will present later, we will be referring to this fixed complete Boolean calculus when we write $\Phi \vdash p$.

In the definition that follows, we will introduce logical formulas for specifying program schemes. We will consider a flexible notion of validity w.r.t. a class of interpretations $\mathcal{C}$. We denote by *All* the class of all nondeterministic interpretations. The class *Det* consists of all deterministic interpretations, that is, interpretations that map atomic letters and program variables to partial functions.

Definition 14 (equations, implications & Hoare formulas)**.** An *equation* is an expression $f \equiv g$, where f and g are program terms. 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 *$\mathcal{C}$-valid*, written $\models f \equiv g$, when every interpretation in $\mathcal{C}$ 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 state space,

$$I, x \models p \text{ and } I(f) : x \mapsto y \text{ imply that } I, y \models q.$$

In this case we write $I \models \{p\}f\{q\}$ and say that I satisfies the Hoare assertion $\{p\}f\{q\}$. 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.

We will use the letter Φ to denote a finite set of tests, and Ψ to denote a finite set of simple Hoare assertions. We say that I satisfies such a collection Ψ, and write $I \models \Psi$, if I satisfies every Hoare assertion in Ψ. We are concerned here with implications of one of the following forms:

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

where Φ is a finite set of tests, and Ψ is a finite set of simple Hoare assertions. We call these formulas *simple implications.* Implications of the last form in particular are called *simple Hoare implications.* An interpretation I satisfies an implication $\Phi, \Psi \Rightarrow \phi$, written $I \models \Phi, \Psi \Rightarrow \phi$, if $I \models \Phi$ and $I \models \Psi$ imply that $I \models \phi$. An implication $\Phi, \Psi \Rightarrow \phi$ is *$\mathcal{C}$-valid*, which we denote by $\Phi, \Psi \models_{\mathcal{C}} \phi$, if every interpretation in $\mathcal{C}$ satisfies it. The implication is said to be *valid* if it is *All*-valid, and we denote this simply by $\Phi, \Psi \models \phi$.

Remark 15 (Hoare assertions as equations)**.** From the definition of satisfaction we get that $I \models \{p\}f\{q\}$ iff the equation $I(p); I(f); {\sim}I(q) = 0$ holds. Now, observe that

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

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.

4 Simple Implicational Theories

In order to cover practical applications, we augment equations $f \equiv g$ between recursion schemes with hypotheses Φ (finite set of tests) and Ψ (finite set of simple Hoare assertions). The main result of this section is that the validity of such an implication $\Phi, \Psi \Rightarrow f \equiv g$ can be reduced to the validity of a simple equation $f^\circ \equiv g^\circ$. 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 a° that in some sense encodes the hypotheses Φ and Ψ.

Note 16. Consider an arbitrary implication $\Phi, \Psi \Rightarrow f \equiv g$. Let $\mathcal{X}$ be the program variables that are free in f or g. Define $\Psi_{\mathcal{X}}$ to be the set that results from Ψ by removing assertions for variables not in $\mathcal{X}$. Let $\mathcal{C}$ be a class of interpretations that satisfies the following closure property: if I is in $\mathcal{C}$, then $I[X \mapsto 0_{SS}]$ is also in $\mathcal{C}$, for any variable X. We can then show easily that $\Phi, \Psi \Rightarrow f \equiv g$ is $\mathcal{C}$-valid iff $\Phi, \Psi_{\mathcal{X}} \Rightarrow f \equiv g$ is $\mathcal{C}$-valid.

$$\begin{aligned} a^\circ \triangleq\ & \mathsf{if}\ \alpha\ \mathsf{then}\ (a; \mathsf{if}\ (q_\alpha \wedge \bigvee \mathsf{At}_\Phi)\ \mathsf{then\ id\ else}\ \bot) \\ & \mathsf{else\ if}\ \beta\ \mathsf{then}\ (a; \mathsf{if}\ (q_\beta \wedge \bigvee \mathsf{At}_\Phi)\ \mathsf{then\ id\ else}\ \bot) \\ & \mathsf{else\ if}\ \dots \\ & \mathsf{else\ if}\ \gamma\ \mathsf{then}\ (a; \mathsf{if}\ (q_\gamma \wedge \bigvee \mathsf{At}_\Phi)\ \mathsf{then\ id\ else}\ \bot) \\ & \mathsf{else}\ \bot, \end{aligned}$$

Figure 2: Encoding the hypotheses Φ and Ψ in the translation a° of an atomic action a.

The previous paragraph means that we can restrict attention w.l.o.g. to implications that do not involve hypotheses for bound variables or for variables that do not appear in the programs. Now, we also observe that free variables can be turned into atomic actions, without changing the interpretation in an essential way.

So, for the results of this section we can assume that the implications $\Phi, \Psi \Rightarrow f \equiv g$ involve hypotheses Ψ only for atomic actions, and additionally f, g are closed (have no free program variables).

Remark 17. Let Φ be a finite set of tests. We observe that $\models \bigwedge \Phi \leftrightarrow \bigvee \mathsf{At}_\Phi$. So, for all intended purposes, the tests $\bigwedge \Phi$ and $\bigvee \mathsf{At}_\Phi$ can be used interchangeably.

Definition 18 (the $(\cdot)^\circ$ transformation)**.** Let Φ and Ψ be finite sets of tests and simple Hoare assertions (for actions) respectively. We fix an atomic action a. For a Φ-consistent atom α we define q_α to be the conjunct

$$q_\alpha \triangleq \bigwedge \{q \mid \{p\}a\{q\} \in \Psi \text{ and } \alpha \leq p\}.$$

Intuitively, q_α is the test that has to hold, according to the hypotheses Ψ, after executing the action a from a state that satisfies the atom α. Define a° as the case statement of Figure 2, where the atoms $\alpha, \beta, \dots, \gamma$ in the statement are an enumeration of the Φ-consistent atoms. We will extend the $(\cdot)^\circ$ transformation to arbitrary programs. First, define the translation of the skip program:

$$\mathsf{id}^\circ \triangleq \mathsf{if}\ (\bigvee \mathsf{At}_\Phi)\ \mathsf{then\ id\ else}\ \bot.$$

Define the substitution $\theta_{\Phi\Psi}$ to map every atomic program a to a°, and the constant id to id°. Finallly, for an arbitrary program term f, we put $f^\circ := \theta_{\Phi\Psi}(f)$.

Lemma 19. Let I be an interpretation that satisfies Φ and Ψ. Then, it holds that $I(a^\circ) = I(a)$ for an atomic action a. In fact, $I(f^\circ) = I(f)$ for every program term f.

Let $A \subseteq S$ and $f : S \rightsquigarrow S$ be a nondeterministic function. We say that f is *A-restricted* if both the domain and the range of f are contained in A:

$$f(x) = \emptyset \text{ for every } x \in S \setminus A \qquad\qquad f(x) \subseteq A \text{ for every } x \in A$$

This means that we can also view f as a nondeterministic function of type $f : A \rightsquigarrow A$.

Observation 20. Let $f, g : S \rightsquigarrow S$ be nondeterministic functions. Let p be a partial function that represents a predicate on S, that is, $p : S \rightharpoonup S$ and $p \leq Id_S$. Consider a subset $A \subseteq S$. If f and g are A-restricted, then so are the functions $f\mathbin{;}g$, $f + g$, and $p\mathbin{;}f$.

Lemma 21 (restriction)**.** Let I be an interpretation, and A be the subset of states that satisfy $\bigvee \mathsf{At}_\Phi$. We assume that for every variable X, the function $I(X)$ is A-restricted. Then, for every program term f, the function $I(f^\circ)$ is A-restricted.

Theorem 22. Let Φ be a finite set of tests, and Ψ be a finite set of simple Hoare assertions for actions. Let $\mathcal{C}$ be either *All* or *Det*. The implication $\Phi, \Psi \Rightarrow f \equiv g$ is $\mathcal{C}$-valid iff the equation $f^\circ \equiv g^\circ$ is $\mathcal{C}$-valid.

Proof. For the right-to-left direction, suppose that the equation $f^\circ \equiv g^\circ$ is $\mathcal{C}$-valid and consider an interpretation I in $\mathcal{C}$ that satisfies Φ and Ψ. We have that $I \models f^\circ \equiv g^\circ$, that is, $I(f^\circ) = I(g^\circ)$. We have to show that I satisfies $f \equiv g$. This follows immediately from Lemma 19, which says that $I(f) = I(f^\circ) = I(g^\circ) = I(g)$.

For the left-to-right direction, suppose that $\Phi, \Psi \Rightarrow f \equiv g$ is $\mathcal{C}$-valid and consider an arbitrary interpretation I in $\mathcal{C}$. We have to show that $I(f^\circ) = I(g^\circ)$. As discussed in Note 16, we can assume w.l.o.g. that the programs f and g have no free program variables, and that no program variable appears in the hypotheses Ψ. Lemma 21 says that $I(f^\circ)$ and $I(g^\circ)$ remain essentially unchanged if we restrict the state space to the states that satisfy $\bigvee \mathsf{At}_\Phi$. So, w.l.o.g. we can assume from now on that $I(\bigvee \mathsf{At}_\Phi) = Id$, that is, I satisfies all the tests in Φ. This means that $I(\mathsf{id}^\circ) = I(\mathsf{id}) = Id$. Now, we want to modify the interpretation function so that every atomic action a is mapped to the nondeterministic function $I(a^\circ)$. We thus define

$$I' = I[a \mapsto I(a^\circ), \text{ for every atomic action } a].$$

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

$$I'(f) = I[a \mapsto I(a^\circ), \text{ for every } a](f) = I(f^\circ)$$

and similarly that $I'(g) = I(g^\circ)$. It follows that $I(f^\circ) = I(g^\circ)$. So, the equation $f^\circ \equiv g^\circ$ is $\mathcal{C}$-valid. ☐

Corollary 23. The simple implicational theory of nondeterministic MRSs (which include the choice operation $+$) is Π^0_1-complete.

Proof. Syntactically, the programs that we consider involve both the recursion operation μ and the nondeterministic choice operation $+$. Semantically, we consider the class *All* of all nondeterministic interpretations. Theorem 22, instantiated with $\mathcal{C}$ being *All*, says that the simple implicational theory of nondeterministic MRSs can be reduced to their equational theory. This gives us the desired Π^0_1 upper bound. Now, Π^0_1-hardness follows trivially from the known Π^0_1-hardness of the equational theory. ☐

Corollary 24. The simple implicational theory of deterministic MRSs (without the $+$ operation) is decidable. In fact, it is a primitive recursive set.

Proof. Syntactically, we consider programs with μ, but without $+$. Semantically, we restrict attention to the class *Det* of deterministic interpretations. Theorem 22, instantiated with $\mathcal{C}$ being *Det*, says that the simple implicational theory of deterministic MRSs can be reduced to their equational theory. 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 a° whose size is proportional to the number of atoms in At_Φ. Decidability follows from the result of Sénizergues on the decidability of the language-equivalence problem for DPDAs [36]. The problems of DPDA equivalence and (strong) equivalence of deterministic MRSs are interreducible, as shown in [15] and [14]. The primitive recursive upper bound follows from the result of Stirling [42], which is a strengthening of the decidability result for DPDA equivalence. □

5 Hoare Calculi for Monadic Recursion Schemes

In this section we propose a Hoare-style calculus (Figure 3) which is sound and complete for the Hoare theory of monadic recursion schemes. The completeness proof will proceed in two steps. First, we will establish in §6 completeness w.r.t. the class *All* of all nondeterministic interpretations. Then, this result will be strengthened in §7 to give us completeness w.r.t. the class *Det* of deterministic interpretations.

We define a proof system in Figure 3 with which we derive Hoare implications. We use the notation $\Phi, \Psi \vdash \{p\}f\{q\}$ to mean that the Hoare implication $\Phi, \Psi \Rightarrow \{p\}f\{q\}$ is provable in our system. In the premise of the μ-rule in Figure 3 appears the notation

$$\Psi[X : \{p_j\}X\{q_j\} \text{ for } j \in K],$$

which denotes the set that results from Ψ by replacing any Hoare assertions for X by the assertions $\{p_j\}X\{q_j\}$ for $j \in K$. The index set K is always taken to be finite. We assume that we have a complete calculus for Boolean logic (see Note 13).

Proposition 25 (soundness)**.** The Hoare-style calculus of Figure 3 is sound for the class *All* of all nondeterministic interpretations.

Proof. Verifying that the rules in Figure 3 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

$$\Psi[X : \{p_j\}X\{q_j\} \text{ for } j \in K].$$

Suppose that $\Phi, \Psi' \models \{p_k\}f\{q_k\}$ for every index $k \in K$. We want to show that

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

for an arbitrary index $\ell \in K$. Let I be an interpretation that satisfies the hypotheses Φ and Ψ. We have to show that $I \models \{p_\ell\}\mu X.f\{q_\ell\}$. Recall the definition for recursion:

$$I(\mu X.f) = \textstyle\sum_{n\geq 0} \tau_n \qquad \tau_0 = 0_{SS} \qquad \tau_{n+1} = I[X \mapsto \tau_n](f)$$

The claim is now that

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

The proof is by induction on n. For the base case, it holds that $I(p_k); \tau_0; {\sim}I(q_k) = 0_{SS}$ because $\tau_0 = 0_{SS}$. For the induction step assume that $I(p_k); \tau_n; {\sim}I(q_k) = 0_{SS}$ for every $k \in K$. Since $I \models \Psi$, we know that I satisfies all the Hoare assertions in Ψ' that do not involve X. Now, notice that for every index $k \in K$:

$$\begin{aligned} I(p_k); \tau_n; {\sim}I(q_k) = 0_{SS} &\iff I(p_k); I[X \mapsto \tau_n](X); {\sim}I(q_k) = 0_{SS} \\ &\iff I[X \mapsto \tau_n] \models \{p_k\}X\{q_k\}. \end{aligned}$$

$$\frac{\{p\}a\{q\} \text{ in } \Psi}{\Phi,\Psi \vdash \{p\}a\{q\}}\ (\mathsf{hyp}) \qquad \frac{\{p\}X\{q\} \text{ in } \Psi}{\Phi,\Psi \vdash \{p\}X\{q\}}\ (\mathsf{hyp}) \qquad \begin{array}{l} \Phi,\Psi \vdash \{p\}\mathsf{id}\{p\}\ (\mathsf{skip}) \\ \Phi,\Psi \vdash \{p\}\bot\{q\}\ (\mathsf{dvrg}) \end{array}$$

$$\frac{\Phi,\Psi \vdash \{p\}f\{q\} \qquad \Phi,\Psi \vdash \{q\}g\{r\}}{\Phi,\Psi \vdash \{p\}f;g\{r\}}\ (\mathsf{seq}) \qquad \frac{\Phi,\Psi \vdash \{p\}f\{q\} \qquad \Phi,\Psi \vdash \{p\}g\{q\}}{\Phi,\Psi \vdash \{p\}f+g\{q\}}\ (\mathsf{ch})$$

$$\frac{\Phi,\Psi \vdash \{p \wedge q\}f\{r\} \qquad \Phi,\Psi \vdash \{\neg p \wedge q\}g\{r\}}{\Phi,\Psi \vdash \{q\}\mathsf{if}\, p\, \mathsf{then}\, f\, \mathsf{else}\, g\{r\}}\ (\mathsf{cond})$$

$$\frac{\Phi,\Psi \vdash \{p \wedge r\}f\{r\}}{\Phi,\Psi \vdash \{r\}\mathsf{while}\, p\, \mathsf{do}\, f\{r \wedge \neg p\}}\ (\mathsf{loop})$$

$$\frac{\Phi,\Psi[X : \{p_j\}X\{q_j\} \text{ for } j \in K] \vdash \{p_k\}f\{q_k\} \text{ for every } k \in K}{\Phi,\Psi \vdash \{p_\ell\}\mu X.f\{q_\ell\}}\ (\mathsf{rec})$$

$$\frac{\Phi \vdash p' \to p \qquad \Phi,\Psi \vdash \{p\}f\{q\} \qquad \Phi \vdash q \to q'}{\Phi,\Psi \vdash \{p'\}f\{q'\}}\ (\mathsf{weak})$$

$$\frac{\Phi,\Psi \vdash \{p\}f\{q_1\} \qquad \Phi,\Psi \vdash \{p\}f\{q_2\}}{\Phi,\Psi \vdash \{p\}f\{q_1 \wedge q_2\}}\ (\mathsf{meet}) \qquad \frac{}{\Phi,\Psi \vdash \{p\}f\{\mathsf{true}\}}\ (\mathsf{meet}_0)$$

$$\frac{\Phi,\Psi \vdash \{p_1\}f\{q\} \qquad \Phi,\Psi \vdash \{p_2\}f\{q\}}{\Phi,\Psi \vdash \{p_1 \vee p_2\}f\{q\}}\ (\mathsf{join}) \qquad \frac{}{\Phi,\Psi \vdash \{\mathsf{false}\}f\{q\}}\ (\mathsf{join}_0)$$

Figure 3: StdHL *(Standard Hoare Logic)*: Proof system for deriving Hoare implications.

It follows that $I[X \mapsto \tau_n]$ satisfies Φ and Ψ', and therefore the premise of the rule gives us that $I[X \mapsto \tau_n] \models \{p_k\}f\{q_k\}$ for every $k \in K$. So, for every $k \in K$:

$$\begin{aligned} I[X \mapsto \tau_n] \models \{p_k\}f\{q_k\} &\iff I(p_k); I[X \mapsto \tau_n](f); I(q_k) = 0_{SS} \\ &\iff I(p_k); \tau_{n+1}; {\sim}I(q_k) = 0_{SS}. \end{aligned}$$

Using the claim we have just proved we can show that

$$I(p_\ell); I(\mu X.f); I(q_\ell) = I(p_\ell);\ \left(\textstyle\sum_{n \geq 0} \tau_n\right); I(q_\ell) = \textstyle\sum_{n \geq 0} I(p_\ell); \tau_n; I(q_\ell) = 0_{SS}.$$

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 (rec) for recursive procedures. 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 and Ψ augmented with the induction hypothesis: for every j in the finite set K, it holds that $\{p_j\}X\{q_j\}$. We show that every part of the claim is preserved:

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

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

```
isEven ≜ if (n = 0) then                isOdd ≜ if (n = 0) then
             out := tt                               out := ff
         else // n > 0                           else // n > 0
             n := n − 1; isOdd                       n := n − 1; isEven
```

Specification for isEven :
$\{even(n)\}\mathsf{isEven}\{out = \mathsf{tt}\}$
$\{\neg even(n)\}\mathsf{isEven}\{out \neq \mathsf{tt}\}$

Atomic tests :	Atomic actions :
$n = 0$	$n := n - 1$
$even(n)$	$out := \mathsf{tt}$
$out = \mathsf{tt}$	$out := \mathsf{ff}$

Hypotheses Φ : $n = 0 \to even(n)$

Hypotheses Ψ : $\{\mathsf{true}\}out := \mathsf{tt}\{out = \mathsf{tt}\}$ $\quad\{n \neq 0 \wedge even(n)\}n := n - 1\{\neg even(n)\}$
$\{\mathsf{true}\}out := \mathsf{ff}\{out \neq \mathsf{tt}\}$ $\quad\{\neg even(n)\}n := n - 1\{even(n)\}$

Figure 4: Mutually recursive procedures isEven and isOdd. The program isEven (resp., isOdd) returns boolean output tt iff the input value n is even (resp., odd).

5.1 A Simple Example

We will now illustrate the Hoare-style calculus of Figure 3 on the simple example program of Figure 4, which consists of the mutually recursive procedures isEven and isOdd. The state space consists of two variables n and out. We think of n as being the input variable, which takes values over the natural numbers. The variable out is used for storing the output, which can be either tt or ff. Our verification task is to establish the following Hoare assertions:

$$\{even(n)\}\mathsf{isEven}\{out = \mathsf{tt}\} \qquad \{\neg even(n)\}\mathsf{isEven}\{out \neq \mathsf{tt}\}$$

These assertions constitute the *specification* for the procedure isEven, and we are asked to show adherence of isEven to this specification.

The first observation we make is that the claim has to be strengthened, so that the necessary properties of the procedure isOdd are also taken into account in the proof. Define the set of simple Hoare assertions *Hyp* to contain the specification for isEven, as well the following assertions:

$$\{even(n)\}\mathsf{isOdd}\{out \neq \mathsf{tt}\} \qquad \{\neg even(n)\}\mathsf{isOdd}\{out = \mathsf{tt}\}$$

The above assertions constitute the specification for the procedure isOdd. By the recursion rule of our Hoare calculus, it suffices to establish the following judgments:

$$\Phi, \Psi, Hyp \vdash \{even(n)\}(n = 0)[out := \mathsf{tt}, (n := n - 1); \mathsf{isOdd}]\{out = \mathsf{tt}\}$$
$$\Phi, \Psi, Hyp \vdash \{\neg even(n)\}(n = 0)[out := \mathsf{tt}, (n := n - 1); \mathsf{isOdd}]\{out \neq \mathsf{tt}\}$$
$$\Phi, \Psi, Hyp \vdash \{even(n)\}(n = 0)[out := \mathsf{ff}, (n := n - 1); \mathsf{isEven}]\{out \neq \mathsf{tt}\}$$
$$\Phi, \Psi, Hyp \vdash \{\neg even(n)\}(n = 0)[out := \mathsf{ff}, (n := n - 1); \mathsf{isEven}]\{out = \mathsf{tt}\}$$

Now, we prove the first of the above judgments. We are reasoning under hypotheses Φ, Ψ, and *Hyp*. We omit them in the proof below, in order to reduce the notational clutter.

1. $n = 0 \wedge even(n) \to \mathsf{true}$ [bool]

2. $\{\mathsf{true}\}out := \mathsf{tt}\{out = \mathsf{tt}\}$	[in Ψ]
3. $\{n = 0 \wedge even(n)\}out := \mathsf{tt}\{out = \mathsf{tt}\}$	[1, 2, weak]
4. $\{n \neq 0 \wedge even(n)\}n := n - 1\{\neg even(n)\}$	[in Ψ]
5. $\{\neg even(n)\}\mathsf{isOdd}\{out = \mathsf{tt}\}$	[in Hyp]
6. $\{n \neq 0 \wedge even(n)\}(n := n - 1); \mathsf{isOdd}\{out = \mathsf{tt}\}$	[4, 5, seq]
7. $\{even(n)\}(n = 0)[out := \mathsf{tt}, (n := n - 1); \mathsf{isOdd}]\{out = \mathsf{tt}\}$	[3, 6, cond]

The proofs for the rest of the obligations are similar and we omit them.

6 First Completeness Theorem

For our first completeness result, we consider the theory of the class *All* of all nondeterministic interpretations. In order to show that our Hoare calculus (Figure 3) is complete, we we will construct a "free" nondeterministic interpretation whose theory is exactly the set of valid (i.e., *All*-valid) Hoare implications. Such an interpretation is called "free" because it is free of extra properties: if an implication is satisfied in it, then it is satisfied in every interpretation.

Let us give an outline of the main ideas for the proof. We consider a finite set Φ of tests, and a finite set Ψ of simple Hoare assertions. The sets Φ and Ψ contain the *hypotheses* under which we want to reason. These hypotheses are meant to constrain the meaning of the primitive letters (atomic tests and atomic actions). We will see how to construct a nondeterministic interpretation $I_{\Phi\Psi}$, which depends on the hypotheses Φ and Ψ. The interpretation $I_{\Phi\Psi}$ satisfies the following properties:

(1) $I_{\Phi\Psi}$ satisfies every test in Φ and every assertion in Ψ.

(2) For a test p, if $I_{\Phi\Psi} \models p$ then $\Phi \vdash p$.

(3) For a Hoare assertion $\{p\}f\{q\}$, if $I_{\Phi\Psi} \models \{p\}f\{q\}$ then $\Phi, \Psi \vdash \{p\}f\{q\}$.

The existence of such an interpretation, which we call a *free interpretation*, has as an easy consequence the completeness of the calculus of Figure 3, as we will see later.

Definition 26 (the free nondeterministic interpretation)**.** Let Φ and Ψ be finite sets of tests and simple Hoare assertions respectively. We define the *free nondeterministic interpretation* $I_{\Phi\Psi}$ (w.r.t. Φ and Ψ) to have At_Φ as state space, and to interpret the tests as I_Φ (the free test interpretation w.r.t. Φ, see Definition 12) does. Moreover, the interpretation $I_{\Phi\Psi}(a) : \mathsf{At}_\Phi \rightsquigarrow \mathsf{At}_\Phi$ of the atomic action a is given by:

$$I_{\Phi\Psi}(a)(\alpha) \triangleq \{\beta \in \mathsf{At}_\Phi \mid \text{for every } \{p\}a\{q\} \text{ in } \Psi.\ \alpha \leq p \implies \beta \leq q\}$$

for every atom α in At_Φ. The program variables $X, Y, \ldots$ are interpreted similarly.

We think that the free nondeterministic interpretation $I_{\Phi\Psi}$ encodes an action-labeled reachability relation on the Φ-consistent atoms At_Φ. If $I_{\Phi\Psi}(a) : \alpha \mapsto \beta$, we say that the atom β is *reachable* from an atom α via the program a (according to the hypotheses of Ψ). This corresponds to the idea that a state satisfying α can be transformed to a state satisfying β when the action a is executed. We extend the notion of reachable atoms to arbitrary programs according to Definition 3. So, the nondeterministic map $I_{\Phi\Psi}(f) : \mathsf{At}_\Phi \rightsquigarrow \mathsf{At}_\Phi$

sends an atom α to the set $I_{\Phi\Psi}(f)(\alpha)$ of atoms that are *reachable via* f. For a conditional $p[f,g]$, we expand the definition:

$$I_{\Phi\Psi}(p[f,g])(\alpha) = \begin{cases} I_{\Phi\Psi}(f)(\alpha), & \text{if } \alpha \leq p \\ I_{\Phi\Psi}(g)(\alpha), & \text{if } \alpha \leq \neg p \end{cases}$$

for every Φ-consistent atom α.

Lemma 27. The free nondeterministic interpretation $I_{\Phi\Psi}$ satisfies both Φ and Ψ.

Observation 28 (functions $\mathsf{At}_\Phi \rightsquigarrow \mathsf{At}_\Phi$ as specifications)**.** Observe that a nondeterministic function $\tau : \mathsf{At}_\Phi \rightsquigarrow \mathsf{At}_\Phi$ is a finite object, and we can think of it as a detailed syntactic specification of the input-output behavior of a program. Intuitively, the function τ *specifies* the program X, if the following Hoare assertions are satisfied: $\{\alpha\}X\{\bigvee \tau(\alpha)\}$ for every Φ-consistent atom α.

Notation 29. Let Φ be a finite set of tests, and Ψ be a finite set of simple Hoare assertions (of the form $\{p\}a\{q\}$ or $\{p\}X\{q\}$). Fix a program variable X and a nondeterministic function $\tau : \mathsf{At}_\Phi \rightsquigarrow \mathsf{At}_\Phi$. We denote by $\Psi[X : \tau]$ the set that results from Ψ by removing all assertions involving X and replacing them by the assertions saying that τ specifies X. That is, we put the assertions $\{\alpha\}X\{\bigvee \tau(\alpha)\}$ for every $\alpha \in \mathsf{At}_\Phi$.

It is easy to see that the free nondeterministic interpretation $I_{\Phi\Psi[X:\tau]}$ is equal to the modified free interpretation $I_{\Phi\Psi}[X \mapsto \tau]$.

Theorem 30 (completeness)**.** Let Φ and Ψ be finite sets of tests and simple Hoare assertions respectively. For every $\alpha \in \mathsf{At}_\Phi$ and every program f, it holds that

$$\Phi, \Psi \vdash \{\alpha\}f\{\textstyle\bigvee I_{\Phi\Psi}(f)(\alpha)\}.$$

Proof. We note that the Hoare assertion $\{\alpha\}f\{\bigvee I_{\Phi\Psi}(f)(\alpha)\}$ is well-formed, because the set of atoms $I_{\Phi\Psi}(f)(\alpha)$ is finite. The proof proceeds by induction on the structure of the program term. For the skip program we have that

$$\frac{\dfrac{}{\Phi, \Psi \vdash \{\alpha\}\mathsf{id}\{\alpha\}}\;(\mathsf{skip}) \qquad \bigvee I_{\Phi\Psi}(\mathsf{id})(\alpha) = \bigvee\{\alpha\} = \alpha}{\Phi, \Psi \vdash \{\alpha\}\mathsf{id}\{\bigvee I_{\Phi\Psi}(\mathsf{id})(\alpha)\}}\;(\text{same assertion}).$$

For the always-diverging program $\bot$, we have the trivial derivation

$$\frac{}{\Phi, \Psi \vdash \{\alpha\}\bot\{\bigvee I_{\Phi\Psi}(\bot)(\alpha)\}}\;(\mathsf{dvrg}).$$

For an atomic program a, we define $\Psi(\alpha, a)$ to be the following set of postconditions:

$$\Psi(\alpha, a) \triangleq \{\text{test } q \mid \text{there is } p \text{ s.t. } \{p\}a\{q\} \in \Psi \text{ and } \alpha \leq p\}.$$

Recall the definition $I_{\Phi\Psi}(a)(\alpha) = \{\beta \in \mathsf{At}_\Phi \mid \forall\{p\}a\{q\} \in \Psi.\ \alpha \leq p \Rightarrow \beta \leq q\}$. We have

$$I_{\Phi\Psi}(a)(\alpha) = \{\beta \in \mathsf{At}_\Phi \mid \forall q \in \Psi(\alpha, a).\ \beta \leq q\} = \textstyle\bigcap_{q \in \Psi(\alpha,a)}\{\beta \in \mathsf{At}_\Phi \mid \beta \leq q\}.$$

Using the fact that $\Phi \models q \leftrightarrow \bigvee\{\beta \in \mathsf{At}_\Phi \mid \beta \leq q\}$ for every test q, we obtain that

$$\Phi \models \textstyle\bigwedge \Psi(\alpha, a) \leftrightarrow \bigwedge_{q \in \Psi(\alpha,a)} q \leftrightarrow \bigwedge_{q \in \Psi(\alpha,a)} \bigvee\{\beta \in \mathsf{At}_\Phi \mid \beta \leq q\} \leftrightarrow$$
$$\textstyle\bigvee \bigcap_{q \in \Psi(\alpha,a)}\{\beta \in \mathsf{At}_\Phi \mid \beta \leq q\} \leftrightarrow \bigvee I_{\Phi\Psi}(a)(\alpha).$$

In the following derivation, we use iterated applications of the (meet) rule to obtain:

$$\dfrac{\dfrac{\Phi \vdash \alpha \to p \qquad \dfrac{\{p\}a\{q\} \text{ in } \Psi}{\Phi,\Psi \vdash \{p\}a\{q\}}\,(\mathsf{hyp})}{\Phi,\Psi\vdash\{\alpha\}a\{q\}}\,(\mathsf{weak}) \qquad \begin{array}{l}\text{for every } \{p\}a\{q\}\\ \text{in } \Psi \text{ with } \alpha \le p\end{array}}{(1)\quad \Phi,\Psi \vdash \{\alpha\}a\{\bigwedge\Psi(\alpha,a)\}}\,(\mathsf{meet})$$

$$\dfrac{(1) \qquad \Phi \vdash \bigwedge\Psi(\alpha,a) \leftrightarrow \bigvee I_{\Phi\Psi}(a)(\alpha)}{\Phi,\Psi\vdash\{\alpha\}a\{\bigvee I_{\Phi\Psi}(a)(\alpha)\}}\,(\mathsf{weak})$$

We handle the base case of a program variable X in an analogous way. We consider now the case of the composite $f;g$. First, we suppose that $I_{\Phi\Psi}(f)(\alpha)$ is empty.

$$\dfrac{\dfrac{\bigvee I_{\Phi\Psi}(f)(\alpha) = \bigvee\emptyset = \mathsf{false}}{\Phi,\Psi\vdash\{\alpha\}f\{\mathsf{false}\}}\,(\text{I.H.}) \qquad \dfrac{}{\Phi,\Psi\vdash\{\mathsf{false}\}g\{\bigvee I_{\Phi\Psi}(f;g)(\alpha)\}}\,(\mathsf{join}_0)}{\Phi,\Psi\vdash\{\alpha\}f;g\{\bigvee I_{\Phi\Psi}(f;g)(\alpha)\}}\,(\mathsf{seq}).$$

We assume now that $I_{\Phi\Psi}(f)(\alpha)$ is nonempty. For an atom β in $I_{\Phi\Psi}(f)(\alpha)$ we obtain:

$$\dfrac{\dfrac{}{\Phi,\Psi\vdash\{\beta\}g\{\bigvee I_{\Phi\Psi}(g)(\beta)\}}\,(\text{I.H.}) \qquad \begin{array}{l}\Phi\vdash \bigvee I_{\Phi\Psi}(g)(\beta)\to\\ \qquad \bigvee\bigcup_{\beta\in I_{\Phi\Psi}(f)(\alpha)} I_{\Phi\Psi}(g)(\beta)\end{array}}{(2)\quad \Phi,\Psi\vdash\{\beta\}g\{\bigvee I_{\Phi\Psi}(f;g)(\alpha)\}}\,(\mathsf{weak})$$

$$\dfrac{\dfrac{}{\Phi,\Psi\vdash\{\beta\}g\{\bigvee I_{\Phi\Psi}(f;g)(\alpha)\}}\,(2) \qquad \begin{array}{l}\text{for every } \beta\\ \text{in } I_{\Phi\Psi}(f)(\alpha)\end{array}}{(3)\quad \Phi,\Psi\vdash\{\bigvee I_{\Phi\Psi}(f)(\alpha)\}g\{\bigvee I_{\Phi\Psi}(f;g)(\alpha)\}}\,(\mathsf{join})$$

$$\dfrac{\dfrac{}{\Phi,\Psi\vdash\{\alpha\}f\{\bigvee I_{\Phi\Psi}(f)(\alpha)\}}\,(\text{I.H.}) \qquad \dfrac{}{\Phi,\Psi\vdash\{\bigvee I_{\Phi\Psi}(f)(\alpha)\}g\{\bigvee I_{\Phi\Psi}(f;g)(\alpha)\}}\,(3)}{\Phi,\Psi\vdash\{\alpha\}f;g\{\bigvee I_{\Phi\Psi}(f;g)(\alpha)\}}\,(\mathsf{seq})$$

For the case of the conditional $p[f,g]$, we first suppose that $\alpha \le p$. The definition says that $I_{\Phi\Psi}(p[f,g])(\alpha) = I_{\Phi\Psi}(f)(\alpha)$. We thus get the following derivation:

1.	$\Phi \vdash p \wedge \alpha \to \alpha$	[prop.]
2.	$\Phi,\Psi \vdash \{\alpha\}f\{\bigvee I_{\Phi\Psi}(f)(\alpha)\}$	[I.H.]
3.	$\Phi,\Psi \vdash \{p\wedge\alpha\}f\{\bigvee I_{\Phi\Psi}(f)(\alpha)\}$	[1, 2, weak]
4.	$\Phi \vdash \neg p \wedge \alpha \to \mathsf{false}$	[$\alpha \le p$, prop.]
5.	$\Phi,\Psi \vdash \{\mathsf{false}\}g\{\bigvee I_{\Phi\Psi}(f)(\alpha)\}$	[join_0]
6.	$\Phi,\Psi \vdash \{\neg p\wedge\alpha\}g\{\bigvee I_{\Phi\Psi}(f)(\alpha)\}$	[4, 5, weak]
7.	$\Phi,\Psi \vdash \{\alpha\}p[f,g]\{\bigvee I_{\Phi\Psi}(f)(\alpha)\}$	[3, 6, cond]
8.	$\Phi,\Psi \vdash \{\alpha\}p[f,g]\{\bigvee I_{\Phi\Psi}(p[f,g])(\alpha)\}$	[same as 7]

The case of $\alpha \le \neg p$ is handled similarly and we omit it.

For the nondeterministic choice $f+g$ we have: $I_{\Phi\Psi}(f+g)(\alpha) = I_{\Phi\Psi}(f)(\alpha) \cup I_{\Phi\Psi}(g)(\alpha)$. So, we obtain the following derivation:

1.	$\Phi,\Psi \vdash \{\alpha\}f\{\bigvee I_{\Phi\Psi}(f)(\alpha)\}$	[I.H.]

2.	$\Phi \vdash \bigvee I_{\Phi\Psi}(f)(\alpha) \to \bigvee I_{\Phi\Psi}(f+g)(\alpha)$	[prop.]
3.	$\Phi, \Psi \vdash \{\alpha\} f \{\bigvee I_{\Phi\Psi}(f+g)(\alpha)\}$	[1, 2, weak]
4.	$\Phi, \Psi \vdash \{\alpha\} g \{\bigvee I_{\Phi\Psi}(f+g)(\alpha)\}$	[similar to 3]
5.	$\Phi, \Psi \vdash \{\alpha\} f+g \{\bigvee I_{\Phi\Psi}(f+g)(\alpha)\}$	[3, 4, ch]

For a loop $\mathsf{w}pf$, we recall the characterization of the meaning of loops as fixpoints. Fix a Φ-consistent atom α. From Observation 4, we have that:

$$I_{\Phi\Psi}(\mathsf{w}pf) = I_\Phi(p); I_{\Phi\Psi}(f); I_{\Phi\Psi}(\mathsf{w}pf) + {\sim} I_\Phi(p).$$

Let $A = I_{\Phi\Psi}(\mathsf{w}pf)(\alpha)$, and we define the set R of *A-safe atoms* as follows:

$$R \triangleq \{\beta \in \mathsf{At}_\Phi \mid I_{\Phi\Psi}(\mathsf{w}pf)(\beta) \subseteq A\}.$$

That is, R is the set of atoms from which we can guarantee the desired postcondition. We show in the claim below that R is an appropriate loop invariant. In fact, it is the weakest (largest) loop invariant that we can consider.

Claim 31. Let β be a Φ-consistent atom with $\beta \in R$. The following hold:

1. *Invariance property*: If $\beta \leq p$, then $I_{\Phi\Psi}(f)(\beta) \subseteq R$.
2. *Safety property*: If $\beta \leq \neg p$, then $\beta \in A$.

The second part of the claim says equivalently that $\{\beta \in R \mid \beta \leq \neg p\} \subseteq A$.

Proof. We show part (1). For the sake of contradiction, we assume that there is some Φ-consistent atom γ in $I_{\Phi\Psi}(f)(\beta)$ with $\gamma \notin R$. Since $\gamma \notin R$, we have that $I_{\Phi\Psi}(\mathsf{w}pf)(\gamma) \not\subseteq A$. The fixpoint equation then implies that

$$I_{\Phi\Psi}(\mathsf{w}pf)(\beta) = [I_{\Phi\Psi}(f); I_{\Phi\Psi}(\mathsf{w}pf)](\beta) = \textstyle\bigcup_{\delta \in I_{\Phi\Psi}(f)(\beta)} I_{\Phi\Psi}(\mathsf{w}pf)(\delta) \not\subseteq A,$$

because $\gamma \in I_{\Phi\Psi}(f)(\beta)$ and $I_{\Phi\Psi}(\mathsf{w}pf)(\gamma) \not\subseteq A$. This contradicts the fact that $\beta \in R$.

For part (2), the fixpoint equation gives us that $I_{\Phi\Psi}(\mathsf{w}pf)(\beta) = \{\beta\}$, because $\beta \leq \neg p$. Since $\beta \in R$, we conclude that $\{\beta\} \subseteq A$, i.e., $\beta \in A$. □

From the definition of R, we have immediately that $\alpha \in R$. For every atom $\beta \in R$ with $\beta \leq p$, we have the following derivation:

1.	$\Phi, \Psi \vdash \{\beta\} f \{\bigvee I_{\Phi\Psi}(f)(\beta)\}$	[I.H.]
2.	$\Phi \vdash \bigvee I_{\Phi\Psi}(f)(\beta) \to \bigvee R$	[Claim 31(1), prop.]
3.	$\Phi, \Psi \vdash \{\beta\} f \{\bigvee R\}$	[1, 2, weak]

Using the above derivation, instantiated for every atom β in $\{\beta \in R \mid \beta \leq p\}$, we get:

4.	$\Phi \vdash \alpha \to \bigvee R$	[$\alpha \in R$, prop.]
5.	$\Phi, \Psi \vdash \{\bigvee\{\beta \in R \mid \beta \leq p\}\} f \{\bigvee R\}$	[3, for all $\beta \in R$ with $\beta \leq p$, join]
6.	$\Phi \vdash (\bigvee R) \wedge p \to \bigvee\{\beta \in R \mid \beta \leq p\}$	[prop.]
7.	$\Phi, \Psi \vdash \{(\bigvee R) \wedge p\} f \{\bigvee R\}$	[5, 6, weak]
8.	$\Phi, \Psi \vdash \{\bigvee R\} \mathsf{w}pf \{(\bigvee R) \wedge \neg p\}$	[7, loop]

9. $\Phi \vdash (\bigvee R) \wedge \neg p \to \bigvee A$ [Claim 31(2), prop.]
10. $\Phi, \Psi \vdash \{\alpha\}\mathsf{w}pf\{\bigvee A\}$ [4, 8, 9, weak]

The last judgment above is the desired $\Phi, \Psi \vdash \{\alpha\}\mathsf{w}pf\{\bigvee I_{\Phi\Psi}(\mathsf{w}pf)(\alpha)\}$.

It remains to consider the case $\mu X.f$ of recursion. Let γ be an arbitrary Φ-consistent atom. We want to show that $\Phi, \Psi \vdash \{\gamma\}\mu X.f\{\bigvee \tau(\gamma)\}$, where $\tau = I_{\Phi\Psi}(\mu X.f)$. Informally, we strengthen the claim to consider all atoms as preconditions. We put

$$\Psi' \triangleq \Psi[X : \tau] = \Psi[X : \{\alpha\}X\{\bigvee \tau(\alpha)\} \text{ for } \alpha \in \mathsf{At}_\Phi].$$

By the (rec) rule for recursion, it suffices to prove that $\Phi, \Psi' \vdash \{\beta\}f\{\bigvee \tau(\beta)\}$ for every atom $\beta \in \mathsf{At}_\Phi$. The fixpoint characterization of recursion in Observation 7 gives us the equation $\tau = I_{\Phi\Psi}[X \mapsto \tau](f)$. As we have already discussed in Notation 29, it holds that $I_{\Phi\Psi}[X \mapsto \tau] = I_{\Phi\Psi[X:\tau]} = I_{\Phi\Psi'}$. Therefore, $\tau = I_{\Phi\Psi'}(f)$. Now,

$$\dfrac{\dfrac{}{\Phi, \Psi' \vdash \{\beta\}f\{\bigvee I_{\Phi\Psi'}(f)(\beta)\}}\,(\text{I.H.}) \qquad \tau = I_{\Phi\Psi'}(f)}{\Phi, \Psi' \vdash \{\beta\}f\{\bigvee \tau(\beta)\}}$$

for every Φ-consistent atom β. This concludes the proof of the theorem. $\square$

Most of the technical work for obtaining completeness of the calculus for the Hoare theory of the class *All* has been done in Theorem 30. Now, we put everything together and give several characterizations of the Hoare theory of *All*.

Corollary 32 (completeness for *All*)**.** Let Φ be a finite set of tests, and Ψ be a finite set of simple Hoare assertions. The following are equivalent:

1. $\Phi, \Psi \models \{p\}f\{q\}$, which is the same as $\Phi, \Psi \models_{All} \{p\}f\{q\}$.
2. $I_{\Phi\Psi} \models \{p\}f\{q\}$. Equivalently, this says that $I_{\Phi\Psi}(f)(\alpha) \subseteq \{\beta \in \mathsf{At}_\Phi \mid \beta \leq q\}$ for every Φ-consistent atom α with $\alpha \leq p$.
3. $\Phi, \Psi \vdash \{p\}f\{q\}$.

Proof. The implication (1) $\Rightarrow$ (2) holds because the interpretation $I_{\Phi\Psi}$ satisfies the hypotheses Φ and Ψ (Lemma 27). For the implication (2) $\Rightarrow$ (3) we have:

$$\dfrac{\dfrac{\dfrac{\text{Theorem 30}}{\Phi, \Psi \vdash \{\alpha\}f\{\bigvee I_{\Phi\Psi}(f)(\alpha)\}} \qquad \dfrac{I_{\Phi\Psi}(f)(\alpha) \subseteq \{\beta \in \mathsf{At}_\Phi \mid \beta \leq q\}}{\Phi \vdash \bigvee I_{\Phi\Psi}(f)(\alpha) \to q}}{\Phi, \Psi \vdash \{\alpha\}f\{q\}}\,(\mathsf{weak}) \qquad \begin{array}{l}\text{for all}\\ \alpha \in \mathsf{At}_\Phi\\ \text{with}\\ \alpha \leq p\end{array}}{\Phi, \Psi \vdash \{\bigvee\{\alpha \in \mathsf{At}_\Phi \mid \alpha \leq p\}\}f\{q\}}\,(\mathsf{join}).$$

Since $\Phi \vdash p \to \bigvee\{\alpha \in \mathsf{At}_\Phi \mid \alpha \leq p\}$, we conclude by ($\mathsf{weak}$) that $\Phi, \Psi \vdash \{p\}f\{q\}$. Finally, the implication (3) $\Rightarrow$ (1) is the soundness result of Proposition 25. $\square$

We observe that the proof of completeness goes through essentially unchanged when we remove arbitrary recursion (with the μ operation) or nondeterministic choice $+$ or both of them. So, we also obtain completeness results for syntactic fragments of the Hoare theory (over *All*) of nondeterministic MRSs. These completeness results are summarized in Figure 5.

syntactic fragment	complete Hoare-style calculus
with μ, with $+$	StdHL (calculus of Figure 3)
with μ, without $+$	StdHL without rule (ch)
without μ, with $+$	StdHL without rule (rec)
without μ, without $+$	StdHL without rules (rec) or (ch)

Figure 5: Four completeness results for syntactic restrictions of nondeterministic MRSs.

The completeness results of Theorem 30 and Corollary 32 give us an effective procedure for the construction of proofs. The free nondeterministic interpretation $I_{\Phi\Psi}$ of Definition 26 is a finite object that can be computed. Then, an inspection of the completeness proof reveals that it constitutes an algorithm that creates a proof out of $I_{\Phi\Psi}$.

A remark is in order for the completeness result shown in the third line of Figure 5. This corollary is closely related to the completeness theorem of Kozen and Tiuryn [31] for the Propositional Hoare Logic of regular programs. Notice, however, that our setting here is much more general, because we can also account for mutual recursion. The result of Kozen of Tiuryn concerns only iteration, not general recursion. Another important difference with the work [31] is that, in the next section, we will strengthen all the completeness results listed in Figure 5 to the case of deterministic interpretations. Showing completeness for a smaller class of interpretations is, of course, a stronger result.

7 Second Completeness Theorem

The completeness theorem of §6 (Corollary 32) is for the Hoare theory of the class *All* of all interpretations. A subclass of *All* that is of particular interest is the class *Det* of deterministic interpretations, which interpret the atomic actions and variables as partial functions. This is a very natural subclass to consider, since we often think of the atomic programs $a, b, \ldots$ as being abstractions of concrete statements, e.g. $x := 3$ or $x := 2 \cdot y + z$, that involve no nondeterminism.

So, the question arises of whether this strictly smaller class of interpretations has the same Hoare theory as *All*. We will show in this section that the Hoare theory of *Det* is in fact *the same as* the Hoare theory of *All*. This is a strengthening of the completeness result of the previous section, and we need that weaker completeness theorem (Corollary 32) as part of the proof.

The idea for the proof is the construction of a free deterministic interpretation $J_{\Phi\Psi}$, which intuitively carries the same information as the free nondeterministic interpretation $I_{\Phi\Psi}$ (Definition 26). We can view the construction of $J_{\Phi\Psi}$ as resolving the abstraction nondeterminism of $I_{\Phi\Psi}$ by blowing-up the state space. The hypotheses Ψ allow nondeterminism in the interpretation of the atomic letters, because the assertion language is abstract and gives only a partial description of the meaning of the letters. The state space of $J_{\Phi\Psi}$ contains enough states to resolve this abstraction nondeterminism. In some sense, a state of $J_{\Phi\Psi}$ specifies the future of the computation and resolves the choices.

Definition 33 (the free deterministic interpretation)**.** Fix finite sets Φ and Ψ of tests and simple Hoare assertions respectively. We define the state space of the *free deterministic* *interpretation* for Φ, Ψ to be the set $\mathsf{At}_\Phi{}^+$ of all finite non-empty strings over the set At_Φ 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 p as follows:

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

We need to consider now the interpretation of the atomic actions and of the program variables. The Hoare assumptions in Ψ restrict the atoms that are *reachable* via an action a or X. The free nondeterministic interpretation $I_{\Phi\Psi}$ carries this information. So, $I_{\Phi\Psi}(a) : \alpha \mapsto \beta$ means that β can be reached from α via a under the restriction that the assumptions Ψ are satisfied. For an atomic program a we define:

$$J_{\Phi\Psi}(a)(\alpha) \triangleq \emptyset \qquad\qquad J_{\Phi\Psi}(a)(\alpha\beta x) \triangleq \begin{cases} \{\beta x\}, & \text{if } I_{\Phi\Psi}(a) : \alpha \mapsto \beta; \\ \emptyset, & \text{otherwise.} \end{cases}$$

Notice that $J_{\Phi\Psi}(a)(\alpha) = \emptyset$, because a single-atom state α signifies that the computation should have terminated. For a program variable X, we define $J_{\Phi\Psi}(X)$ analogously.

Lemma 34. The free deterministic interpretation $J_{\Phi\Psi}$ satisfies Φ and Ψ.

Definition 35 (strongest postconditions)**.** For a state space S, a nondeterministic function $\phi : S \rightsquigarrow S$ and a predicate $P \subseteq S$, we define

$$\mathbf{post}(P, \phi) \triangleq \textstyle\bigcup_{x \in P} \phi(x) = \bigcup\{\phi(x) \mid x \in P\}.$$

For the particular case where the state space is a set $S = A^+$ of nonempty strings, for some nonempty set A, we have that

$$\mathbf{post}(\alpha \cdot A^*, \phi) = \bigcup\{\phi(\alpha x) \mid x \in A^*\},$$

where α is an element of A.

Definition 36 (agreement for functions)**.** Consider a state space A, and the state space A^+ of non-empty finite words over A. We say that the function $\phi : A^+ \rightsquigarrow A^+$ *agrees with* the function $\psi : A \rightsquigarrow A$ if the following equation holds for every $\alpha \in A$:

$$\mathbf{post}(\alpha \cdot A^*, \phi) = \psi(\alpha) \cdot A^*,$$

where A^* is the set of finite words over A. Intuitively, agreement of ϕ with ψ says that we can calculate strongest postconditions for ϕ using the function ψ instead.

Lemma 37 (agreement)**.** Let A be a state space, and consider the families of nondeterministic functions $\phi_i : A^+ \rightsquigarrow A^+$ and $\psi_i : A \rightsquigarrow A$. The following hold:

1. If ϕ_1, ϕ_2 agree with ψ_1, ψ_2 respectively, then $\phi_1 \mathbin{;} \phi_2$ agrees with $\phi_1 \mathbin{;} \psi_2$.
2. If ϕ_i agrees with ψ_i for every $i \in K$, then $\sum_{i \in K} \phi_i$ agrees with $\sum_{i \in K} \psi_i$.

The index set K above can be of arbitrary cardinality.

Definition 38 (agreement for interpretations)**.** Let A be a state space, I be an interpretation over A, and J be an interpretation over A^+. We say that J *agrees with* I *on the test* p if the following holds:

$$J(p)(\alpha x) = I(p)(\alpha) \cdot x \quad \text{for every state } \alpha x \in A^+.$$

Equivalently, the equation above says that $J, \alpha x \models p$ iff $I, \alpha \models p$. For a program term f, we say that J *agrees with* I *on* f if the function $J(f) : A^+ \rightsquigarrow A^+$ agrees with the function $I(f) : A \rightsquigarrow A$ (Definition 36). Finally, we say that J *agrees with* I if they agree on every (atomic) test and on every program term.

Proposition 39 (agreement)**.** Let A be a state space, I be an interpretation over A, and J be an interpretation over A^+. Suppose that J agrees with I on tests, as well as on atomic actions a and program variables X. Then,

$$\mathbf{post}(\alpha \cdot A^*, J(f)) = I(f)(\alpha) \cdot A^*$$

for every $\alpha \in A$ and every program term f. That is, J agrees with I on all programs.

Proof. The proof is by induction on the structure of the program term f. We will only show here the case $\mu X.f$ of recursion, which is the most interesting case. We recall the definition of the interpretation under I and J.

$$\begin{array}{lll} I(\mu X.f) = \sum_{n\geq 0} \sigma_n & \sigma_0 = 0_{AA} & \sigma_{n+1} = I[X \mapsto \sigma_n](f) \\ J(\mu X.f) = \sum_{n\geq 0} \tau_n & \tau_0 = 0_{A^+A^+} & \tau_{n+1} = J[X \mapsto \tau_n](f) \end{array}$$

The types of the recursion approximants are $\sigma_n : A \rightsquigarrow A$ and $\tau_n : A^+ \rightsquigarrow A^+$. Now, we need to establish the following auxiliary claim:

Claim 40. For every $n \geq 0$, The function τ_n agrees with σ_n .

Proof. For the base case $n = 0$, we observe that $0_{A^+A^+}$ agrees with 0_{AA}. For the step, we want to show that τ_{n+1} agrees with σ_{n+1}. The inner I.H. says that τ_n agrees with σ_n, which implies that the modified interpretations $I[X \mapsto \sigma_n]$ and $J[X \mapsto \tau_n]$ satisfy the assumptions of the proposition. Then, we invoke the outer I.H. to obtain that $J[X \mapsto \tau_n]$ agrees with $I[X \mapsto \sigma_n]$ on f. It follows that τ_{n+1} agrees with σ_{n+1}, which concludes the proof of the claim. ☐

Using the above claim gives and Part (2) of Lemma 37, we conclude that $J(\mu X.f)$ agrees with $I(\mu X.f)$. ☐

Theorem 41 (agreement of $J_{\Phi\Psi}$ with $I_{\Phi\Psi}$)**.** Let Φ and Ψ be finite sets of tests and simple Hoare assertions respectively. Then, the free deterministic interpretation $J_{\Phi\Psi}$ agrees with the free nondeterministic interpretation $I_{\Phi\Psi}$.

Proof. Recall that the state space of $I_{\Phi\Psi}$ is At_Φ (Definition 26), and the state space of $J_{\Phi\Psi}$ is ${\mathsf{At}_\Phi}^+$ (Definition 33). By virtue of Proposition 39, it suffices to show:

1. $J_{\Phi\Psi}$ agrees with $I_{\Phi\Psi}$ on all (atomic) tests.

2. $J_{\Phi\Psi}$ agrees with $I_{\Phi\Psi}$ on every atomic program a and every program variable X.

Obligation (1) is easy to establish: $J_{\Phi\Psi}(p)(\alpha x) = I_{\Phi\Psi}(p)(\alpha) \cdot x$ for every atomic test p, and every state $\alpha x \in \mathsf{At}_\Phi{}^+$, according to Definition 33. Agreement on atomic tests then implies agreement on all tests. For obligation (2), we consider an arbitrary action a and an atom $\alpha \in \mathsf{At}_\Phi$. It holds that:

$$\begin{aligned}
\mathbf{post}(\alpha \cdot \mathsf{At}_\Phi{}^*, J_{\Phi\Psi}(a)) &= \bigcup\{J_{\Phi\Psi}(a)(\alpha x) \mid x \in \mathsf{At}_\Phi{}^*\} \\
&= \bigcup\{J_{\Phi\Psi}(a)(\alpha\beta x) \mid x \in \mathsf{At}_\Phi{}^*\} \\
&= \bigcup\{\{\beta x\} \mid I_{\Phi\Psi}(a) : \alpha \mapsto \beta,\ x \in \mathsf{At}_\Phi{}^*\} \\
&= \{\beta x \mid I_{\Phi\Psi}(a) : \alpha \mapsto \beta,\ x \in \mathsf{At}_\Phi{}^*\} \\
&= \{\beta \mid I_{\Phi\Psi}(a) : \alpha \mapsto \beta\} \cdot \mathsf{At}_\Phi{}^* \\
&= I_{\Phi\Psi}(a)(\alpha) \cdot \mathsf{At}_\Phi{}^*.
\end{aligned}$$

So, $J_{\Phi\Psi}(a)$ agrees with $I_{\Phi\Psi}(a)$. The proof is analogous for the case of a variable X. □

Corollary 42 (completeness for *Det*)**.** Let Φ and Ψ be finite sets of tests and simple Hoare assertions respectively. The following are equivalent:

(1) $\Phi, \Psi \models_{Det} \{p\}f\{q\}$.

(2) $J_{\Phi\Psi} \models \{p\}f\{q\}$.

(3) $I_{\Phi\Psi}(f)(\alpha) \subseteq \{\beta \in \mathsf{At}_\Phi \mid \beta \leq q\}$ for every atom $\alpha \in \mathsf{At}_\Phi$ with $\alpha \leq p$.

(4) $\Phi, \Psi \vdash \{p\}f\{q\}$.

Proof. We show (1) $\Rightarrow$ (2). $J_{\Phi\Psi}$ is the free deterministic interpretation for Φ, Ψ (see Definition 33). We know from Lemma 34 that $J_{\Phi\Psi}$ satisfies both Φ and Ψ, and $J_{\Phi\Psi}$ belongs to the class *Det* of deterministic interpretations. From the hypothesis $\Phi, \Psi \models_{Det} \{p\}f\{q\}$ we then obtain that $J_{\Phi\Psi} \models \{p\}f\{q\}$.

Now, we show (2) $\Rightarrow$ (3). Let α be a Φ-consistent atom with $\alpha \leq p$. In order to show the containment $I_{\Phi\Psi}(f)(\alpha) \subseteq \{\beta \in \mathsf{At}_\Phi \mid \beta \leq q\}$, we consider an atom $\beta \in I_{\Phi\Psi}(f)(\alpha)$. In different notation, $I_{\Phi\Psi}(f) : \alpha \mapsto \beta$. From the "agreement of $J_{\Phi\Psi}$ with $I_{\Phi\Psi}$" result of Theorem 41, we get that the function $J_{\Phi\Psi}(f) : \mathsf{At}_\Phi{}^+ \rightsquigarrow \mathsf{At}_\Phi{}^+$ agrees with the function $I_{\Phi\Psi}(f) : \mathsf{At}_\Phi \rightsquigarrow \mathsf{At}_\Phi$. This means that

$$\mathbf{post}(\alpha \cdot \mathsf{At}_\Phi{}^*, J_{\Phi\Psi}(f)) = \bigcup\{J_{\Phi\Psi}(f)(\alpha x) \mid x \in \mathsf{At}_\Phi{}^*\} = I_{\Phi\Psi}(f)(\alpha) \cdot \mathsf{At}_\Phi{}^*.$$

Since the atom β is in $I_{\Phi\Psi}(f)(\alpha) \cdot \mathsf{At}^*$, the above equation implies that there exists some word $x \in \mathsf{At}_\Phi{}^*$ such that $\beta \in J_{\Phi\Psi}(f)(\alpha x)$. That is, $J_{\Phi\Psi}(f) : \alpha x \mapsto \beta$. Since $J_{\Phi\Psi}, \alpha x \models p$, the assumption $J_{\Phi\Psi} \models \{p\}f\{q\}$ gives us that $J_{\Phi\Psi}, \beta \models q$. It follows that $\beta \leq q$. We have thus proved the desired containment.

The implication (3) $\Rightarrow$ (4) has already been proved in Corollary 32. Finally, the implication (4) $\Rightarrow$ (3) follows from the soundness result of Proposition 25, which establishes $\Phi, \Psi \models_{All} \{p\}f\{q\}$, and the fact that the class *Det* is contained in *All*, which gives us the desired $\Phi, \Psi \models_{Det} \{p\}f\{q\}$. □

As in §6, the completeness result for the Hoare theory of *Det*, which is the same as the Hoare theory of *All*, extends to syntactic restrictions without μ or without $+$. So, the results that are summarized in Figure 5 still apply.

8 Complexity of the Hoare theory

We investigate here the computational complexity of the problem μHoare: "Given finite sets Φ, Ψ of tests and simple Hoare assertions respectively, and a Hoare assertion $\{p\}f\{q\}$, is it the case that $\Phi, \Psi \vdash \{p\}f\{q\}$?". In order find an algorithmic solution to this problem, we make use of our completeness theorems. In particular, we consider the auxiliary characterization of the Hoare theory that they provide in terms of the free nondeterministic interpretation $I_{\Phi\Psi}$.

Theorem 43. The problem μHoare is in $\mathsf{EXPTIME}$.

Proof. The statement $\Phi, \Psi \vdash \{p\}f\{q\}$ is equivalent to: $\Phi, \Psi \vdash \{\alpha\}f\{q\}$ for every atom $\alpha \in \mathsf{At}_\Phi$ with $\alpha \leq p$. It follows that we can restrict attention to Hoare consequences of the special form $\{\alpha\}f\{q\}$.

As we showed in Corollary 32, the statement $\Phi, \Psi \vdash \{\alpha\}f\{q\}$, where α is an atom in At_Φ, is equivalent to the containment

$$I_{\Phi\Psi}(f)(\alpha) \subseteq \{\beta \in \mathsf{At}_\Phi \mid \beta \leq q\},$$

where $I_{\Phi\Psi}$ is the free nondeterministic interpretation (Definition 26). So, the problem amounts to computing the interpretation $I_{\Phi\Psi}(f) : \mathsf{At}_\Phi \rightsquigarrow \mathsf{At}_\Phi$ for arbitrary programs f. We will give a procedure for computing an explicit representation of $I_{\Phi\Psi}(f)$.

Let k be the number of atomic tests that appear in the input Φ, Ψ, $\{p\}f\{q\}$. The size N of the set At_Φ of Φ-consistent atoms is bounded above by 2^k, which is equal to 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 nondeterministic function $\mathsf{At}_\Phi \rightsquigarrow \mathsf{At}_\Phi$ as a $\mathsf{At}_\Phi \times \mathsf{At}_\Phi$ 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 $O(N^2)$. Moreover, the sum operation $+$ on matrices is given by: $(\sigma + \tau)_{\alpha\beta} = \sigma_{\alpha\beta} + \tau_{\alpha\beta}$. Computing the sum $\sigma + \tau$ also takes time $O(N^2)$. Since while loops $\mathsf{w}pf$ can be encoded using recursion $\mu X.p[f; X, \mathsf{id}]$ (see Claim 8), we do not need to deal with them here.

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:

$$\begin{array}{lll} PS(b, \bar{\sigma}) \triangleq \sigma_b & PS(\mathsf{id}, \bar{\sigma}) \triangleq \mathbf{1}_{\mathsf{At}_\Phi} & PS(g; h, \bar{\sigma}) \triangleq PS(g, \bar{\sigma})\,\mathbf{;}\, PS(h, \bar{\sigma}) \\ PS(Y, \bar{\sigma}) \triangleq \sigma_Y & PS(\bot, \bar{\sigma}) \triangleq \mathbf{0}_{\mathsf{At}_\Phi} & PS(p[g, h], \bar{\sigma}) \triangleq p[PS(g, \bar{\sigma}), PS(h, \bar{\sigma})] \\ & & PS(g + h, \bar{\sigma}) \triangleq PS(g, \bar{\sigma}) + PS(h, \bar{\sigma}) \end{array}$$

where $\mathbf{0}_{\mathsf{At}_\Phi}$ is the matrix with 0's everywhere, and $\mathbf{1}_{\mathsf{At}_\Phi}$ has 1's on the diagonal and 0's elsewhere. We describe the case $\mu Y.g$ of recursion in a more operational way in Figure 6.

```
$PS(\mu Y.g, \bar{\sigma}) \triangleq \{$
    $S := \mathbf{0}_{\mathsf{At}_\Phi}$
    for $t = 1, \ldots, N \cdot N$ do
        $newS := PS(g, \bar{\sigma}[Y \mapsto S])$
        $S := newS$
    return $S$
$\}$
```

Figure 6: Definition of the recursive procedure $PS(f, \bar{\sigma})$ for the case $f = \mu Y.g$.

We write $\bar{\sigma}[Y \mapsto S]$ to denote the modification of $\bar{\sigma}$ that maps Y to the matrix S. It is straightforward to see that

$$PS(f, \bar{\sigma}) = I_{\Phi\Psi}(f), \quad \text{where } \sigma_a = I_{\Phi\Psi}(a) \text{ and } \sigma_X = I_{\Phi\Psi}(X).$$

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 and Ψ can clearly be done in exponential time. $\square$

Theorem 44. The problem μHoare is $\mathsf{EXPTIME}$-hard.

Proof. We show how to encode the computations of polynomial-space bounded alternating Turing machines [7]. 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 $\sqcup$, 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

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

```
$Y[q,i,a] \triangleq$ 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 accepted
        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 rejected
        id                           // propagate failure upwards
    }
```

Figure 7: Encoding universal branching with recursive calls.

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 $Y[q,i,a]$:

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

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 want to take the following assumptions:

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

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 7. 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 8). 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:

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

```
$Y[q,i,a] \triangleq$ 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 $(\neg A)$ 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 8: Encoding existential branching with recursive calls.

syntactic fragment	complexity of Hoare theory
with μ, with +	EXPTIME-complete [32]
with μ, without +	EXPTIME-complete [32]
without μ, with +	PSPACE-complete [27]
without μ, without +	PSPACE-complete [27]

Figure 9: Complexity of the Hoare theory for several syntactic fragments.

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

$$start = P_1^{x_1} \wedge \cdots \wedge P_n^{x_n} \wedge P_{n+1}^{\llcorner\!\lrcorner} \wedge \cdots \wedge P_{\pi(n)}^{\llcorner\!\lrcorner},$$

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 the input string $x_1 x_2 \ldots x_n$ iff

$$\Phi, \Psi \vdash \{start\} X[q_0, 1] \{A\},$$

where Φ is the collection of our assumptions for the atomic tests, and Ψ is the collection of assumptions for the atomic programs. Showing this claim involves using the characterization of the Hoare theory in terms of $I_{\Phi\Psi}$ (Corollary 32). $\square$

The proof of the EXPTIME upper bound accounts for program schemes that involve the recursion μ and the nondeterministic choice + operations. For the lower bound of EXPTIME-hardness, however, the presented encoding does not use the + operation. We thus obtain EXPTIME-completeness for the +-free fragment as well. See Figure 9. The first two results for EXPTIME-completeness are proved here and in the conference version of the present work [32]. The two last results were presented in [27, 28, 8].

9 Summary & Conclusion

In conclusion, we reiterate the remark that reasoning about mutually recursive programs presents some fundamental difficulties. We encounter this difficulty even at the abstract propositional level: the equational theory of CFGs is Π^0_1-complete, and therefore it is not amenable to an effective axiomatization. On the positive side, the equational theory of deterministic MRSs is decidable, but the best known algorithm for deciding this theory is nonelementary.

In order to accommodate program verification applications, we need to reason under hypotheses that capture useful properties of the domain of computation. Towards this end, we showed that the simple implicational theory of MRSs, which allows hypotheses of the form $\{p\}a\{q\}$, reduces to their equational theory. The reduction incurs an exponential blow-up.

In search of more computationally manageable fragments, we considered the class of properties that can be expressed as Hoare assertions $\{p\}f\{q\}$ under extra hypotheses. For this subclass of properties, we saw that we can also handle nondeterminism without increasing the complexity or complicating the axiomatization.

We investigated the propositional Hoare theory of deterministic and nondeterministic MRSs, and we obtained sound and complete Hoare-style calculi. Our completeness results are unconditional (not relative completeness in the sense of Cook [9]). We believe that the proposed axiomatization is intuitive, and it lends itself both to the manual and automatic construction of proofs.

Finally, the Hoare theory of MRSs was shown to be EXPTIME-complete. This implies, in particular, that the proofs of valid Hoare implications can also be constructed in exponential time.

Building on ideas of the present work, the propositional Hoare theory of *dual nondeterminism* is investigated in [33]. An abstract programming language with while loops is considered, where two different kinds of nondeterminism are allowed: *angelic* and *demonic*. In this setting, the meaning of a program is a game between the angel and the demon, where the angel tries to satisfy the specification and the demon tries to falsify it. Two unconditional completeness results are obtained for the so-called *weak* and *strong Hoare theory* of dual nondeterminism. The computational complexity of the Hoare theory is investigated using operational models that correspond to safety games. Finally, a sound and complete Hoare-style calculus for synthesizing angelic strategies is presented. This provides a deductive approach for the synthesis of programs.

References

[1] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: Semantic foundations for networks. In *Proceedings of the 41st annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2014)*, pages 113–126, 2014.

[2] Krzysztof R. Apt. Ten years of Hoare's logic: A survey – Part I. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 3(4):431–483, 1981.

[3] Krzysztof R. Apt. Ten years of Hoare's logic: A survey – Part II: Nondeterminism. *Theoretical Computer Science*, 28(1):83–109, 1983.

[4] Stanislav Böhm and Stefan 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.

[5] Stanislav Böhm, Stefan Göller, and Petr 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.

[6] Stanislav Böhm, Stefan Göller, and Petr Jancar. Equivalence of deterministic one-counter automata is NL-complete. *CoRR*, abs/1301.2181, 2013.

[7] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

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

[9] Stephen A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, 1978.

[10] Martin Davis. *Computability and Unsolvability.* Dover Publications, 1982.

[11] Michael J. Fischer and Richard E. Ladner. Propositional modal logic of programs. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing (STOC '77)*, pages 286–294, 1977.

[12] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[13] Robert W. Floyd. Assigning meanings to programs. In *Mathematical Aspects of Computer Science, Proceedings of AMS Symposium in Applied Mathematics*, volume 19, pages 19–32, 1967.

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

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

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

[17] Niels Bjørn Bugge Grathwohl, Fritz Henglein, and Dexter Kozen. Infinitary axiomatization of the equational theory of context-free languages. *EPTCS*, 126:44–55, 2013.

[18] Niels Bjørn Bugge Grathwohl, Dexter Kozen, and Konstantinos Mamouras. KAT + B! In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, pages 44:1–44:10, 2014.

[19] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic.* MIT Press, 2000.

[20] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580,583, 1969.

[21] Petr Jančar. A short decidability proof for DPDA language equivalence via first-order grammars. *CoRR*, abs/1010.4760, 2011.

[22] Petr 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.

[23] Dexter 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.

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

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

[26] Dexter Kozen. Typed Kleene algebra. Technical report, Cornell University, 1998.

[27] Dexter 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.

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

[29] Dexter Kozen and Konstantinos Mamouras. Kleene algebra with products and iteration theories. In *Proceedings of the 22nd EACSL Annual Conference on Computer Science Logic (CSL 2013)*, pages 415–431, 2013.

[30] Dexter Kozen and Konstantinos Mamouras. Kleene algebra with equations. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP 2014)*, pages 280–292, 2014.

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

[32] Konstantinos Mamouras. On the Hoare theory of monadic recursion schemes. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, pages 69:1–69:10, 2014.

[33] Konstantinos Mamouras. Synthesis of strategies and the Hoare logic of angelic nondeterminism. 2014. To be presented at FoSSaCS 2015, London, UK.

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

[35] Vaughan R. Pratt. Semantical considerations on Floyd-Hoare logic. In *Proceedings of the 17th IEEE Annual Symposium on Foundations of Computer Science (FOCS 1976)*, pages 109–121, 1976.

[36] Géraud 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.

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

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

[39] Géraud 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.

[40] Michael Sipser. *Introduction to the Theory of Computation.* Cengage Learning, 2nd edition, 2005.

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

[42] Colin 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.

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