Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 228 (2009) 101–112
www.elsevier.com/locate/entcs

Inductive Completeness of Logics of Programs

Daniel Leivant

Indiana University, Bloomington

Abstract

We propose a new approach to delineating logics of programs, based directly on inductive definition of program semantics. The ingredients are elementary and well-known, but their fusion yields a simple yet powerful approach, surprisingly overlooked for decades.
The denotational semantics of a regular program can be construed as a relation, easily definable by structural induction on programs. Invoking the framework of canonical theories for (iterated) inductive definitions, we consider the first-order theory for program semantic, i.e. with the generative clauses as construction (introduction) rules, and their dual templates as deconstruction (elimination) rules.
We prove that Hoare's logic is *inductively complete,* in the sense that a partial-correctness assertion is Hoare provable iff it is provable in the inductive theory (with deconstruction for formulas in the base vocabulary). Thus first-order automated theorem-proving can be applied directly to program verification.
Proceeding to program termination, we show that a total correctness assertion is valid iff it is provable in the inductive theory without any use of deconstruction. This is yet another take on the first-order nature of total correctness.

Keywords: Inductive definitions, program semantics, inductive completeness, Hoare logic

1 Introduction

1.1 Background

Logics of programs inherently exceed first-order logic, because program semantics is defined in terms of iterative processes. We can capture such definitions using second-order formulas [9], existential fixpoints [1], or explicit reference to the natural numbers (The Hungarian School), but not by a first-order theory. Consequently, the delineation of logics of programs cannot parallel the characterization of first-order logic by soundness and completeness for general ("uninterpreted") validity.

The early attempts to refer instead to *local completeness,* i.e. completeness with respect to one structure at a time, led to Cook's Relative Completeness Theorem. However, notwithstanding the persistent centrality of Cook's relative completeness [2] in PL research, that notion has foundational and practical drawbacks [10]. For one, relative completeness fails to demarcate the boundaries of logics of programs:

[1] leivant@cs.indiana.edu

doi:10.1016/j.entcs.2008.12.119

there are, for example, proper extensions of Hoare's Logic that are sound and relatively complete!

An alternative approach is to match logics of programs with formalisms for explicit reasoning about programs, such as second-order logic. A common objection to second-order logic, that it is non-axiomatizable for its intended semantics, is off the mark here. On the one hand there are natural deductive calculi for second-order logic, which are sound and complete for natural non-standard semantics (Henkin's structures); on the other hand, the same objection applies to arithmetic! Indeed, it turns out that Hoare's Logic matches second-order logic with first-order set-existence, and Dynamic Logic matches second-order logic with "computational" (i.e. strict-Π^1_1) set-existence [9,10].

Here we consider an approach dual to explicit second-order rendition of program semantics, namely an *implicit* but *first-order* rendition. We consider the inductive definition of program semantics, and invoke the well-known framework of first-order theories for such definitions [7,8,4,12,5]. We show that Hoare's Logic matches the inductive theory with deconstruction (i.e. induction) restricted to the base vocabulary. Also, the valid total correctness assertions match with the generative fragment of the inductive theory (i.e. with no deconstruction at all).

Since this approach is strictly first-order, it is particularly accessible conceptually, expositorily, and for use of automated theorem proving tools. It also meshes generically with the long-standing tradition of defining program semantics inductively. As soon as an inductive definition is given for a programming language, we automatically obtain the corresponding first-order inductive-definition theory, and can tackle the question of completeness of a logic for that theory.

A match between Hoare's logic and inductive theories was observed already by the Hungarian school of "nonstandard dynamic logic" (see e.g. [3,16]). However, the first-order theories they considered invoke the natural numbers as an auxiliary data-type. Our present approach calls for for no such extraneous data-types, and is more generic, in that it applies directly to all programming constructs with inductively defined semantics, even when they lack a simple iterative definition in terms of the natural numbers.

1.2 Guarded iterative programs

To focus on the essentials, we consider guarded iterative programs, i.e. regular programs with tests and assignments as atomic actions. This provides a clean separation between the basic programming concepts. The language is generic with respect to an underlying vocabulary V, consisting of a finite set of constant-, function- and relation-identifiers, assigned positive arities when appropriate.

As usual, *V-assignments* are expressions of the form $x := \mathbf{t}$, where x is a variable and $\mathbf{t}$ a V-term. In addition to assignment, we use as atomic programs *queries*. of the form $?\varphi$, φ a first-order V-formula. In practice queries are restricted to quantifier-free formulas, but this makes no difference for our discussion. The *regular V-programs* $\alpha, \beta, \ldots$ are obtained from atomic programs by composition $(\alpha; \beta)$, union $\alpha \cup \beta$, and nondeterministic iteration (α^*).

Given a V-structure $\mathcal{S}$, the operational semantics of programs is obvious. Recall (e.g. from [6]) that "while" programs are definable in terms of guarded iterative programs. Namely, a program `if` φ `then` α `else` β `endif` is rendered by $(?\varphi;\alpha) \cup (?\neg\varphi;\varphi)$; and `while` φ `do` α `enddo` is rendered by $(?\varphi;\alpha)^*;?\neg\varphi$.

1.3 Inductive definition of program semantics

Generic methods for associating to a given collection of inductive (i.e. generative) definitions first-order inductive theories are well-known. The semantics of regular programs has a particularly simple form of inductive definition, using atomic production rules, i.e. natural-deduction inferences with atomic premises and conclusion, as follows.

For a list $\boldsymbol{x} = (x_1, \ldots, x_n)$ of variables, let $\mathbf{P}[\boldsymbol{x}]$ consists of the regular V-programs with all assigned variables among $x_1 \ldots x_n$. Note that if α is such a program, then so are all its subprograms. Given a V-structure $\mathcal{S}$, each program $\alpha \in \mathbf{P}[\boldsymbol{x}]$ defines a $2n$-ary relation $[\![\alpha]\!]_{\mathcal{S}}$ on the universe $|\mathcal{S}|$ of $\mathcal{S}$, that holds between $\boldsymbol{a}, \boldsymbol{b} \in |\mathcal{S}|^n$ iff α has a complete execution that starts with $\boldsymbol{x}$ bound to $\boldsymbol{a}$, and terminates with $\boldsymbol{x}$ bound to $\boldsymbol{b}$.

For $n \geq 1$, let $\hat{V}^n$ be the expansion of the underlying vocabulary V with $2n$-ary relational identifiers $\mathtt{M}^n_\alpha$ for each $\alpha \in \mathbf{P}[\boldsymbol{x}]$. The intent is that $\mathtt{M}^n_\alpha$ denotes, in each V-structure $\mathcal{S}$, the relation $[\![\alpha]\!]^n$ above. We omit n when in no danger of confusion.

An inductive definition of $[\![\alpha]\!]^n$, uniform for all V-structures, is given by generative clauses that can be rendered by following atomic rule-templates, which can be construed as natural-deduction rules.

ASSIGNMENT $\dfrac{}{\mathtt{M}_{x_i:=\mathbf{t}}(\boldsymbol{x}, \boldsymbol{x}_{i\leftarrow t})}$ where $\boldsymbol{x}_{i\leftarrow t}$ is $x_1 \ldots x_{i-1}, \mathbf{t}, x_{i+1} \ldots x_n$

TEST $\dfrac{\varphi[\boldsymbol{x}]}{\mathtt{M}_{?\varphi}(\boldsymbol{x}, \boldsymbol{x})}$

COMPOSITION $\dfrac{\mathtt{M}_\beta(\boldsymbol{x}, \boldsymbol{u}) \quad \mathtt{M}_\gamma(\boldsymbol{u}, \boldsymbol{v})}{\mathtt{M}_{\beta;\gamma}(\boldsymbol{x}, \boldsymbol{v})}$

BRANCHING $\dfrac{\mathtt{M}_\beta(\boldsymbol{x}, \boldsymbol{v})}{\mathtt{M}_{\beta\cup\gamma}(\boldsymbol{x}, \boldsymbol{v})} \qquad \dfrac{\mathtt{M}_\gamma(\boldsymbol{x}, \boldsymbol{v})}{\mathtt{M}_{\beta\cup\gamma}(\boldsymbol{x}, \boldsymbol{v})}$

ITERATION $\dfrac{}{\mathtt{M}_{\beta^*}(\boldsymbol{x}, \boldsymbol{x})} \qquad \dfrac{\mathtt{M}_{\beta^*}(\boldsymbol{x}, \boldsymbol{u}) \quad \mathtt{M}_{\beta^*}(\boldsymbol{u}, \boldsymbol{v})}{\mathtt{M}_\beta(\boldsymbol{x}, \boldsymbol{v})}$

1.4 Expressing program properties

We refer to the language of first-order dynamic logic, DL, as e.g. in [6]. It is easy to see that, modulo the intended reading of the identifiers $\mathtt{M}_\alpha$, the expressive power of the $\hat{V}$-formulas is identical to the expressive power of DL formulas over the base vocabulary V. To avoid clutter let us posit that all assigned-variables in programs are among $x_1 \ldots x_n$, and that $\hat{V}$ stands for $\hat{V}^n$. Recall that in a DL formula $\forall u\ \varphi$

the variable u cannot be among the assigned variables in φ, and we can therefore posit that no quantified variables are among $x_1 \dots x_n$.[2]

Each DL V-formula φ can be expressed as a $\hat{V}$-formula $\varphi^\sharp$, defined by structural recurrence on φ. For φ modal-free we take of course $\varphi^\sharp$ to be φ itself. If φ is $[\alpha]\varphi_0$, then $\varphi^\sharp$ is $\forall v_1 \dots v_n\, \mathtt{M}_\alpha(\boldsymbol{x}, \boldsymbol{v}) \to \{\boldsymbol{v}/\boldsymbol{x}\}\varphi_0^\sharp$. Finally, we let $\sharp$ commute with the first-order logical operations; that is: if φ is $\varphi_0 \wedge \varphi_1$ then $\varphi^\sharp$ is $\varphi_0^\sharp \wedge \varphi_1^\sharp$ and similarly for other connectives and the quantifiers.

Conversely, each $\hat{V}$-formula ψ is expressible as a DL V-formula $\psi^\natural$, defined by structural recurrence on ψ. If ψ is a V-formula, we defined $\psi^\natural$ to be ψ. If ψ is $\mathtt{M}_\alpha(\mathbf{t}, \mathbf{q})$ then $\psi^\natural$ is $\boldsymbol{x} = \mathbf{t} \to \langle\alpha\rangle(\boldsymbol{x} = \mathbf{q})$. And $\natural$ commutes with the first-order logical operations.

Observe that these interpretations are trivially *sound* in the following sense.

Theorem 1.1 *For every V-structure $\mathcal{S}$, if $\hat{\mathcal{S}}$ is the $\hat{V}$-expansion of $\mathcal{S}$ in which each $\mathtt{M}_\alpha$ is interpreted as the denotational semantic of α, then for every DL V-formula φ, $\hat{\mathcal{S}} \models \varphi \leftrightarrow \varphi^\sharp$, and for every $\hat{V}$-formula ψ, $\hat{\mathcal{S}} \models \psi \leftrightarrow \psi^\natural$.*

Corollary 1.2 *$\mathcal{S} \models \varphi$ iff $\hat{\mathcal{S}} \models \varphi^\sharp$.*

1.5 The inductive theory of regular programs

The generative rules above, defining inductively program semantics, bound the interpretation of the relation-identifiers $\mathtt{M}_\alpha$ from below. Bounding inductively generated sets from above, namely as the *minimal* relations closed under the generative clauses, is a second-order condition which has no first-order axiomatization (except for degenerated cases). However, we can approximate that delineation as the minimal one among a given collection of *definable* relations. Namely, the deconstruction template for $\mathtt{M}_\alpha$ states that $\mathtt{M}_\alpha$ is contained in every definable relation closed under the generative rules for $\mathtt{M}_\alpha$. A familiar example is the deconstruction for the set $\mathbb{N}$ of natural numbers. With N as unary relational-identifier, the generative clauses are

$$\frac{}{N(\mathtt{0})} \quad \text{and} \quad \frac{N(x)}{N(\mathtt{s}(x))}$$

yielding the Deconstruction template

$$\frac{N(x) \quad \varphi[\mathtt{0}] \quad \begin{array}{c}\varphi[z]\\ \cdots\\ \varphi[\mathtt{s}z]\end{array}}{\varphi[x]} \qquad \begin{array}{l}\text{(assumption } \varphi[z] \text{ is discharged (i.e. closed))}\\ \text{(z not free in other open assumptions}\end{array}$$

that is, the natural-deduction rule of induction on $\mathbb{N}$ [15].

Analogously, the DECONSTRUCTION RULE for the iteration construct $*$ should be

[2] Note that an expression such as $\forall x\ [x := 1](x = 1)$ is not a legal DL formula, whereas $\forall x\ [y := x](y = x)$ is.

$$\dfrac{\mathtt{M}_{\beta^*}(\boldsymbol{x},\boldsymbol{v}) \quad \varphi[\boldsymbol{z},\boldsymbol{z}] \quad \overset{\varphi[\boldsymbol{u},\boldsymbol{w}] \quad \mathtt{M}_\beta(\boldsymbol{z},\boldsymbol{u})}{\cdots} \quad \varphi[\boldsymbol{z},\boldsymbol{w}]}{\varphi[\boldsymbol{x},\boldsymbol{v}]}$$

(assumptions $\mathtt{M}_\beta(\boldsymbol{z},\boldsymbol{u})$, $\varphi[\boldsymbol{u},\boldsymbol{w}]$ are discharged

$\boldsymbol{z},\boldsymbol{w},\boldsymbol{u}$ not free in other open assumptions)

The formula φ above is the *eigen-formula* of the inference.

A related, more practical template is

$$\textsc{Unfolding} \qquad \dfrac{\mathtt{M}_{\beta^*}(\boldsymbol{x},\boldsymbol{v}) \quad \psi[\boldsymbol{u}] \to \psi[\boldsymbol{w}] \quad \overset{\mathtt{M}_\beta(\boldsymbol{u},\boldsymbol{w})}{\cdots}}{\psi[\boldsymbol{x}] \to \psi[\boldsymbol{v}]}$$

However, we have

Proposition 1.3 *The schemas* Deconstruction *and* Unfolding *are equivalent.*

Proof. Posit the Deconstruction schema, and assume the premises of Unfolding. Then the three premises of Deconstruction hold with $\varphi[\boldsymbol{x},\boldsymbol{y}]$ taken as $\psi[\boldsymbol{x}]\to\psi[\boldsymbol{y}]$. We thus obtain $\psi[\boldsymbol{x}]\to\psi[\boldsymbol{v}]$, as required.

Conversely, posit the Unfolding schema, and assume the premises of Deconstruction. Then the premises of Unfolding hold with $\psi[\boldsymbol{y}]$ taken to be $\varphi[\boldsymbol{x},\boldsymbol{y}]$. Thus Unfolding yields $\psi[\boldsymbol{x}] \to \psi[\boldsymbol{v}]$, establishing Deconstruction. □

Note that deconstruction rules for the remaining program constructs are degenerate, in the sense that they are equivalent to explicit definitions. For example, the Deconstruction of composition, combined with Composition Introduction yield an explicit definition of $\mathtt{M}_{\beta;\gamma}$. More generally, $\mathtt{M}_\alpha$ can be explicitly defined in terms of components of α, for all non-loop programs α,

- $\mathtt{M}_{x_i:=t}(\boldsymbol{x},\boldsymbol{v}) \;\leftrightarrow\; (v_i = \mathbf{t}[\boldsymbol{v}] \wedge \bigwedge_{j\neq i} v_j = x_j)$.
- $\mathtt{M}_{?\chi}(\boldsymbol{x},\boldsymbol{v}) \;\leftrightarrow\; (\chi \wedge \boldsymbol{v} = \boldsymbol{x})$
- $\mathtt{M}_{\beta;\gamma}(\boldsymbol{x},\boldsymbol{v}) \leftrightarrow \exists \boldsymbol{u}\, \mathtt{M}_\beta(\boldsymbol{x},\boldsymbol{u}) \;\wedge\; \mathtt{M}_\gamma(\boldsymbol{u},\boldsymbol{v})$
- $\mathtt{M}_{\beta\cup\gamma}(\boldsymbol{x},\boldsymbol{v}) \;\leftrightarrow\; \mathtt{M}_\beta(\boldsymbol{x},\boldsymbol{v}) \;\vee\; \mathtt{M}_\gamma(\boldsymbol{x},\boldsymbol{v})$

We write $\mathbf{Ind}^n(\mathit{Reg})$ for the inductive theory given by the universal closure of the formulas above. We omit the superscript n as well as *Reg* when in no danger of confusion. Two weaker theories are of interest. The *Elementary inductive-theory,* $\mathbf{Ind}_0$ has inductive-elimination restricted to first-order eigen-formulas. The *Generative Theory* **Gen** is weaker yet, and has only the inductive-introduction rules, without inductive-elimination. Thus **Gen** is an inherently first-order theory, in that it does not have templates intended to approximate a second-order rule.

2 Partial correctness

2.1 Hoare Logic for regular programs

Recall that a partial correctness assertion (PCA) can be construed as a DL formula of the form $\varphi \to [\alpha]\,\psi$, abbreviated $\varphi\,[\alpha]\,\psi$, where φ and ψ are (modal-free) first-order. In a partial-correctness logic the only modal formulas used are PCAs.

Given a vocabulary V and a V-theory $\mathbf{T}$, the Hoare's Logic (based on $\mathbf{T}$) for regular programs is the partial-correctness logic axiomatized by the following calculus $\mathbf{H}^*(\mathbf{T})$. Note that the first-order formulas used in $\mathbf{H}^*(\mathbf{T})$ may refer to vocabulary identifiers not appearing in $\mathbf{T}$; in particular, even if $\mathbf{T}$ is empty, all first-order formulas are permissible. Let $\vdash$ denote provability in first-order logic. The inference rules of $\mathbf{H}(\mathbf{T})$ are as follows.

Assignment	$\{\mathbf{t}/x\}\varphi \;\; [x := \mathbf{t}] \;\; \varphi$
Composition	$\dfrac{\psi\ [\alpha]\ \chi \qquad \chi\ [\beta]\ \varphi}{\psi\ [\alpha;\beta]\ \varphi}$
Branching	$\dfrac{\psi\ [\alpha]\ \varphi \qquad \psi\ [\beta]\ \varphi}{\psi\ [\alpha \cup \beta]\ \varphi}$
Iteration	$\dfrac{\varphi\ [\alpha]\ \varphi}{\varphi\ [\alpha^*]\ \varphi}$
Query	$\dfrac{\mathbf{T} \vdash \psi \wedge \chi \to \varphi}{\psi[?\chi]\varphi}$
Consequence	$\dfrac{\mathbf{T} \vdash \psi' \to \psi \qquad \psi\ [\alpha]\ \varphi \qquad \mathbf{T} \vdash \psi' \to \psi}{\psi'\ [\alpha]\ \varphi'}$

A formalism for reasoning about PCAs for guarded iterative programs is obtained by replacing the rules for Branching, Query, and Iteration by rules for the remaining program constructs of guarded iterative programs.

2.2 Inductive soundness of Hoare's Logic

Clearly, Hoare logic is semantically sound. Only slightly less trivial is the observation that it is sound for $\mathbf{Ind}_0$:

Theorem 2.1 *If* $\mathbf{H}^*(\mathbf{T}) \vdash \psi\ [\alpha]\ \varphi$ *then* $\mathbf{T} + \mathbf{Ind}_0 \vdash (\psi\ [\alpha]\ \varphi)^\sharp$.

Proof. Straightforward induction on proofs in $\mathbf{T} + \mathbf{Ind}_0$.

Consider, for example, Hoare's ITERATION Rule. If $\psi\ [\alpha^*]\ \psi$ is provable in $\mathbf{H}^*(\mathbf{T})$, then we posit by IH that

$$\mathbf{T} + \mathbf{Ind}_0 \vdash \psi[\boldsymbol{x}] \wedge \mathtt{M}_\alpha(\boldsymbol{x}, \boldsymbol{v}) \ \rightarrow\ \psi[\boldsymbol{v}]$$

But then we indeed have, by the UNFOLDING Rule

$$\mathbf{T} + \mathbf{Ind}_0 \vdash \varphi[\boldsymbol{x}] \wedge \mathtt{M}_{\alpha^*}(\boldsymbol{x}, \boldsymbol{v}) \ \rightarrow\ \varphi[\boldsymbol{v}]$$ □

A point of interest is that first-order proofs of $\mathbf{T} + \mathbf{Ind}_0$ obtained in the proof of Theorem 2.1 do not use the generative (data introduction) rules of the inductive theory **Ind**. A dual observation holds for total correctness assertions (Theorem 3.1).

2.3 Approximating program semantics with finitely many instances

The second-order nature of the relations $\mathtt{M}_{\beta^*}$ is approximated by the first-order schema of Deconstruction. Moreover, any particular proof $\mathcal{P}$ of $\mathbf{Ind}_0$ uses, for each β^*, only finitely many instances of Deconstruction for $\mathtt{M}_{\beta^*}$. We show here how this can be used.

Let A be a finite collection of programs, with all assigned variables among $x_1 \ldots x_n$. Let Φ be a mapping that assigns to each $\beta^* \in A$ a finite set $\Phi(\beta)$ of $2n$-ary first-order predicates $\lambda \boldsymbol{y}, \boldsymbol{z}\ \varphi_i$. For $\alpha \in A$, let $\mathtt{M}^\Phi_\alpha$ be a first-order formula, defined by induction on α, as follows.

- For atomic α (assignment or test) $\mathtt{M}^\Phi_\alpha$ is the explicit definition of $\mathtt{M}_\alpha$, as above.
- $\mathtt{M}^\Phi_{\beta;\gamma}(\boldsymbol{u}, \boldsymbol{v})$ is $\exists \boldsymbol{w}\ \mathtt{M}^\Phi_\beta(\boldsymbol{u}, \boldsymbol{w}) \wedge \mathtt{M}^\Phi_\gamma(\boldsymbol{w}, \boldsymbol{v})$.
- $\mathtt{M}^\Phi_{\beta\cup\gamma}(\boldsymbol{u}, \boldsymbol{v})$ is $\mathtt{M}^\Phi_\beta(\boldsymbol{u}, \boldsymbol{v}) \vee \mathtt{M}^\Phi_\gamma(\boldsymbol{u}, \boldsymbol{v})$.
- $\mathtt{M}^\Phi_{\beta^*}(\boldsymbol{u}, \boldsymbol{v})$ is

$$\bigwedge \{\ Cl^\Phi[\varphi] \rightarrow \varphi[\boldsymbol{u}, \boldsymbol{v}]\ \mid\ (\lambda \boldsymbol{y}, \boldsymbol{z}.\ \varphi[\boldsymbol{y}, \boldsymbol{z}])\ \in\ \Phi(\beta^*)\ \}$$

where $Cl^\Phi[\varphi]$ is

$$(\ \forall \boldsymbol{w} \varphi^\Phi[\boldsymbol{w}, \boldsymbol{w}]\ )\ \wedge\ (\ \forall \boldsymbol{w}, \boldsymbol{y}, \boldsymbol{z}\ \varphi^\Phi[\boldsymbol{w}, \boldsymbol{y}] \wedge \mathtt{M}^\Phi_\beta(\boldsymbol{y}, \boldsymbol{z}) \rightarrow \varphi^\Phi[\boldsymbol{w}, \boldsymbol{z}]\ )$$

Lemma 2.2 *For $\alpha \in A$ and Φ as above,* $\mathbf{Ind} \vdash \mathtt{M}_\alpha(\boldsymbol{u}, \boldsymbol{v}) \rightarrow \mathtt{M}^\Phi_\alpha(\boldsymbol{u}, \boldsymbol{v})$

Proof. The proof is straightforward by (structural) induction on φ. □

Fix A, Φ as above, and let ψ be a formula not using the variables $\boldsymbol{w}, \boldsymbol{y}, \boldsymbol{z}$ used in defining the predicates $\mathtt{M}^\Phi_\alpha$, and with all programs in A. We write ψ^Φ for the result of replacing in ψ each atomic subformula $\mathtt{M}_\alpha(\mathbf{t}, \mathbf{q})$ by the formula $\mathtt{M}^\Phi_\alpha[\mathbf{t}, \mathbf{q}]$ defined above.

Lemma 2.3 *Suppose $\mathcal{P}$ is a proof in $\mathbf{T} + \mathbf{Ind}$ of a formula ψ. Let $\Phi(\beta)$ be the set of eigen-formulas in $\mathcal{P}$ of Deconstruction for $\mathtt{M}_{\beta^*}$. Then $\mathbf{T} \vdash \psi^\Phi$.*

Proof. The Lemma, generalized from formulas ψ to sequents $\Gamma \Rightarrow \psi$, is proved by a straightforward induction on proofs. □

2.4 Inductive completeness of Hoare's Logic

Our main result about partial correctness logic is the completeness of Hoare's Logic for the Elementary Inductive Theory $\mathbf{Ind}_0$. While this theorem easily implies Cook's Relative Completeness Theorem, we cannot simply adapt Cook's proof. If we could, then the proof would also apply to the full inductive theory **Ind**, for which the Theorem's statement is false: there are PCAs provable in **Ind** that are not provable in Hoare's Logic [10]. Thus, the proof of Theorem 2.1 must depend on the restriction of Deconstruction to first-order formulas.

Theorem 2.4 *For all V-theories* $\mathbf{T}$*, if* $\mathbf{T} + \mathbf{Ind}_0 \vdash (\psi\ [\alpha]\ \varphi)^\sharp$*, then* $\mathbf{H}^*(\mathbf{T}) \vdash \psi\ [\alpha]\ \varphi$*.*

Proof. (Outline). The proof is by (structural) induction on α.

- Suppose α is an assignment, say $x_1 := \mathbf{t}$. Then $(\varphi[\alpha]\psi)^\sharp$ is

$$\forall \boldsymbol{u}, \boldsymbol{v}.\ \varphi[\boldsymbol{u}] \wedge v_1 = \mathbf{t}[\boldsymbol{u}] \wedge \bigwedge_{i=2..k} v_i = u_i \to \psi[\boldsymbol{v}]$$

  which trivially implies

$$\varphi[\boldsymbol{x}] \to \psi[\mathbf{t}[\boldsymbol{x}], x_2 \dots x_k]$$

  Since this formula is valid, it is first-order provable. But in $\mathbf{H}^*$ we have

$$\psi[\mathbf{t}[\boldsymbol{x}], x_2 \dots x_k]\ [\alpha]\ \psi[\boldsymbol{x}]$$

  which combined with the formula above by the rule of Consequence yields $\varphi[\alpha]\psi$.
- α is $?\chi$. Then $(\varphi[\alpha]\psi)^\sharp$ is

$$\forall \boldsymbol{u}, \boldsymbol{v}.\ \varphi[\boldsymbol{u}] \wedge \chi[\boldsymbol{u}] \wedge \boldsymbol{v} = \boldsymbol{u} \ \to\ \psi[\boldsymbol{v}]$$

  and so $\mathbf{T} \vdash_{\mathbf{Ind}} \varphi \wedge \chi \ \to\ \psi$. Since this entailment is valid, we have $\mathbf{T} \vdash_{\mathbf{L}_1} \varphi \wedge \chi \ \to\ \psi$, and so $\varphi[\alpha]\psi$ is obtained in $\mathbf{H}^*$ by the Query Rule.
- α is $\beta;\gamma$. Then $(\varphi[\alpha]\psi)^\sharp$ is

$$\forall \boldsymbol{u}, \boldsymbol{v}.\ \varphi[\boldsymbol{u}] \wedge \exists \boldsymbol{w}.\ \mathtt{M}_\beta[\boldsymbol{u}, \boldsymbol{w}] \wedge \mathtt{M}_\gamma[\boldsymbol{w}, \boldsymbol{v}] \ \to\ \psi[\boldsymbol{v}]$$

  Thus, we assume that $\mathbf{Ind}_0$ proves

$$\mathtt{M}_\beta[\boldsymbol{u}, \boldsymbol{w}] \wedge \mathtt{M}_\gamma[\boldsymbol{w}, \boldsymbol{v}] \ \to\ (\varphi[\boldsymbol{u}] \to \psi[\boldsymbol{v}])$$

  By Lemma 2.3 there are expansions $\mathtt{M}'_\beta[\boldsymbol{u}, \boldsymbol{w}]$ and $\mathtt{M}'_\gamma[\boldsymbol{w}, \boldsymbol{v}]$ of $\mathtt{M}_\beta[\boldsymbol{u}, \boldsymbol{w}]$ and $\mathtt{M}_\gamma[\boldsymbol{w}, \boldsymbol{v}]$ respectively, such that

$$M'_\beta[\boldsymbol{u}, \boldsymbol{w}] \wedge M'_\gamma[\boldsymbol{w}, \boldsymbol{v}] \ \to\ (\varphi[\boldsymbol{u}] \to \psi[\boldsymbol{v}])$$

Let

$$\chi[\boldsymbol{x}] \quad \equiv \quad M'_\gamma[\boldsymbol{x},\boldsymbol{v}] \ \rightarrow \ \psi[\boldsymbol{v}]$$

By Lemma 2.2 we have

$$\mathbf{Ind}_0,\ \mathbf{T} \ \vdash \quad \varphi[\boldsymbol{u}] \wedge M_\beta[\boldsymbol{u},\boldsymbol{w}] \rightarrow \chi[\boldsymbol{w}],$$

so by IH $\mathbf{H}^*(\mathbf{T})$ proves $\varphi[\beta]\chi$. By Lemma 2.2 we also have, in $\mathbf{Ind}_0$,

$$\chi[\boldsymbol{w}] \wedge M_\gamma[\boldsymbol{w},\boldsymbol{v}] \rightarrow \psi[\boldsymbol{v}],$$

which by IH implies that $\mathbf{H}^*(\mathbf{T})$ proves $\chi[\gamma]\psi$. Using the Composition Rule of $\mathbf{H}^*$ we obtain $\varphi[\alpha]\psi$.

- α is $\beta \cup \gamma$. Then $(\varphi[\alpha]\psi)^\sharp$ is

$$\forall \boldsymbol{u},\boldsymbol{v}.\ \varphi[\boldsymbol{u}] \wedge (M_\beta[\boldsymbol{u},\boldsymbol{w}] \vee M_\gamma[\boldsymbol{w},\boldsymbol{v}]) \ \rightarrow \ \psi[\boldsymbol{v}]$$

from which we have both $(\varphi[\beta]\psi)^\sharp$ and $(\varphi[\gamma]\psi)^\sharp$. By IH both $\varphi[\beta]\psi$ and $\varphi[\gamma]\psi$ are provable in $\mathbf{H}^*(\mathbf{T})$, from which $\varphi[\alpha]\psi$ follows by the Branching Rule of $\mathbf{H}^*$.

- α is β^*. Then $(\varphi[\alpha]\psi)^\sharp$ is

$$\varphi[\boldsymbol{u}] \wedge M_{\beta^*}[\boldsymbol{u},\boldsymbol{v}] \ \rightarrow \ \psi[\boldsymbol{v}]$$

By Lemma 2.3 there is a finite set Ξ of first-order predicates such that, in first order logic,

$$\varphi[\boldsymbol{u}] \wedge M^{\Xi}_{\beta^*}[\boldsymbol{u},\boldsymbol{v}] \rightarrow \psi[\boldsymbol{v}] \tag{1}$$

Define

$$\chi[\boldsymbol{x}] \quad \equiv_{\mathrm{df}} \quad \exists \boldsymbol{w}.\varphi[\boldsymbol{w}] \wedge M^{\Xi}_{\beta^*}[\boldsymbol{w},\boldsymbol{x}]$$

By Lemma 2.2 we have $\vdash M_\beta[\boldsymbol{u},\boldsymbol{v}] \rightarrow M^{\Xi}_{\beta}[\boldsymbol{u},\boldsymbol{v}]$, so (1) implies

$$\chi[\boldsymbol{u}] \wedge M_\beta[\boldsymbol{u},\boldsymbol{v}] \rightarrow \chi[\boldsymbol{v}]$$

which by IH implies that $\chi[\beta]\chi$ is provable in $\mathbf{H}^*(\mathbf{T})$. Using the Iteration Rule of $\mathbf{H}^*$, we get $\chi[\beta^*]\chi$.

But in first order logic we have $\varphi[\boldsymbol{x}] \rightarrow \chi[\boldsymbol{x}]$, by taking $\boldsymbol{w} = \boldsymbol{x}$. Using the Rule of Consequence, we thus obtain $\varphi[\beta^*]\chi$ in $\mathbf{H}^*(\mathbf{T})$. On the other hand, by (1) we have $\chi[\boldsymbol{x}] \rightarrow \psi[\boldsymbol{x}]$ in first order logic. So by the rule of Consequence we get $\varphi[\beta^*]\psi$. □

A result related to Theorem 2.4 was proved by Ildiko Sain [16], who showed that "Floyd Method" is complete for truth in all structures where a predicate denoting program-semantics satisfies "relational induction". Sain's treatment differs methodologically from our approach in several ways. First, it refers to Csirmaz's abstract notion of "program" [3] which is based on first-order expressibility of program semantics within a structure, and on provable termination, two requirements

that offset the notion's generality. This means that the relevance of results proved about Csirmaz's programs to Hoare's Logic, not only require a special adaptation, but are limited to terminating programs. Moreover, the approach of [3], while striving for generality, is in fact limited to simple programming constructs, and requires an auxiliary machinery for expressing program semantics within the structures in hand.

2.5 Relative completeness

Relative completeness has been regarded as a canonical bill of health for proposed logics of imperative programs. A first point worth noting is that relative completeness is an immediate consequence of Theorem 2.4. If a V-structure $\mathcal{S}$ is expressive (in the sense of Cook) then each relation $\mathtt{M}_\alpha$ is definable in $\mathcal{S}$ by some V-formula ξ_α. Fixing a program α with assigned variables among $x_1 \dots x_n$, let $\mathbf{Ind}_\xi$ be $\mathbf{Ind}^n$ with each $\mathtt{M}_\beta$ replaced by ξ_β. By Theorem 2.4, if $\mathbf{T}$ is a V-theory, and

$$\mathbf{T}, \mathbf{Ind}_\xi \vdash \varphi[\boldsymbol{x}] \wedge \xi_\alpha[\boldsymbol{x}, \boldsymbol{u}] \to \psi[\boldsymbol{v}] \tag{2}$$

then $\mathbf{T} \vdash_H \varphi[\alpha]\psi$. Taking for $\mathbf{T}$ the complete first-order theory $Th(\mathcal{S})$ of $\mathcal{S}$, (2) is trivial, since the derived formula is an element of $\mathbf{T}$, so $Th(\mathcal{S}) \vdash_H \varphi[\alpha]\psi$.

It thus appears that relative completeness is a "local projection" of inductive completeness, obtained when one examines individual structures rather than all structures, and further requires those structures to be expressive. Inductive completeness is the more general and global property, and it applies to all structures, regardless of expressiveness.

Moreover, the inductive soundness and completeness of Hoare's logic provides an exact delineation: adding PCAs to Hoare's logic dashes inductive soundness, and eliminating a PCA would destroys inductive completeness. Delineation is most intuitive reading of "complete": no proper extension of the logic is sound. This is what Gödel's Completeness Theorem does for classical first-order logic, Kripke's for constructive logic, and Henkin's for classical higher-order logic. In contrast, relative completeness fails to delineate Hoare's logic: adding to Hoare's logic $\mathbf{H}$ any valid PCA unprovable in $\mathbf{H}$ yields a proper extension $\mathbf{H}^+$ which is again sound and relatively complete. Since $\mathbf{H}^+$ is not a natural logic, it follows that relative-completeness fails to explain the naturalness of $\mathbf{H}$.

Relative completeness also lacks the genericity of traditional completeness properties. The completeness of classical first-order logic implies that for any first-order theory $\mathbf{T}$, say the algebraic theory of fields, if $\mathbf{T} \models \varphi$, then $\mathbf{T} \vdash \varphi$. Similarly, inductive completeness implies that if $\mathbf{T} + \mathbf{Ind}_0$ proves a PCA, then the that PCA is provable in $\mathbf{H}$. Relative completeness has no such consequence.

Relative completeness throws in the towel not only regarding structures that are not expressive (and for which inductive completeness is unproblematic), but also for programming languages whose termination problem for finite structures is not decidable. In contrast, giving inductive definitions to the semantics of such programs is straightforward, albeit calling for auxiliary constructs (e.g. stacks). The

study of inductive completeness for such programming languages is therefore also possible, indeed straightforward. We shall pursue this elsewhere.

3 Total correctness and the Generative Theory

Recall that total-correctness assertions (TCA) (over a vocabulary V) are DL formulas of the form $\varphi \to \langle\alpha\rangle\psi$, where φ and ψ are first-order V-formulas. It is well known that the validity of TCAs (in all V-structures!) is fundamentally a first-order property [14]. For example, there is a simple log-space transducer to transform a TCA τ to a first order formula φ_τ so that τ is valid iff φ is [11].

The inductive framework provides yet another take on the first-order nature of TCAs (compare [14,13,17]).

Theorem 3.1 *Let V be a vocabulary, A TCA τ over V is valid iff $\tau^\sharp$ is provable in* **Gen**.

More generally, if $\mathbf{T}$ *a V-theory, then* $\mathbf{T}$ *entails τ (i.e. τ is true in every model of* $\mathbf{T}$*) iff* $\mathbf{T} + \mathbf{Gen} \vdash \tau^\sharp$.

Proof. The validity of provable TCAs is trivial. Conversely, if a TCA τ is valid, then $\tau^\sharp$ is true whenever each M_α is interpreted as the semantics of α. But because M_α appears positively in $\tau^\sharp$, the latter remains true if M_α is extended; thus only the closure conditions on the M_α's matter. □

Since the deconstruction template for an inductive definition is an approximation to a second-order axiom, it is of a rather different nature from the generative axioms. For one, it has infinitely many instances, and cannot be reduced to any finite number of them. Thus the subformula property (a consequence of cut-elimination by which every formula φ has a proof using only subformulas of φ) fails miserably. In contrast, the generative theory is truly first-order, with a finite number of axioms expressible as atomic inference rules.

4 Conclusion

Based on inductive definability as a generic framework for specifying program semantics, we proposed here canonical theories of inductive definability as the point of reference for proving completeness (and soundness) of logics of programs. We illustrated the framework by applying it to regular programs, and to Hoare's logic for them. The completeness results we obtain are simple, direct, and completely general (no restriction to expressive structures).

The semantic of regular programs refers to states of bounded size n (the number of assignable variables used), and so the semantics of programs is expressible as relations of arity $2n$. Less elementary programming constructs, say recursive procedures with local variables, may require inductive definitions referring to auxiliary data objects (such as lists), which is precisely what we have in mind. The effect of this would be particularly obvious for programming languages with undecidable finite-structure halting, for which no relatively complete axiomatizations

exist: the auxiliary constructs take over program complexity; however, we can still define natural logics for such languages, and prove their inductive completeness.

Our first take on completeness for logics of programs, in [9], was based on second-order logic as a universal framework, with the expectation that first-order comprehension corresponds to logics for PCAs, and comprehension for computation (i.e. relational Π^1_1) formulas corresponding to dynamic logic. However, the situation is more subtle. As manifested in [10], different forms of computational formulas correspond to different program constructs, and the second-order framework does not provide a silver bullet that applies to all constructs. The inductive framework studies here is therefore more appealing and generic: the programming construct in had guides directly the inductive definition, and once it does the match with a program logic is obtained.

References

[1] Andreas Blass and Yuri Gurevich. The underlying logic of Hoare logic. *Current Trends in Theoretical Computer Scienc*, pages 409–436, 2001.

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

[3] L. Csirmaz. Programs and program verification in a general setting. *Theoretical Computer Science*, 16:199–210, 1981.

[4] Solomon Feferman. Formal theories for transfinite iterations of generalized inductive definitions and some subsystems of analysis. In *Intuitionism and Proof Theory*, pages 303–326. North-Holland, Amsterdam, 1970.

[5] Solomon Feferman and Wilfried Sieg. Iterated inductive definitions and subsystems of analysis. In *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoreitc Studies*, LNM 897, pages 16–77. Springer-Verlag, Berlin, 1981.

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

[7] G. Kreisel. Generalized inductive definitions. Reports for the seminar on foundations of analysis, Stanford, Volume 1 §3, 1963.

[8] G. Kreisel. Mathematical logic. In T. Saaty, editor, *Lectures on Modern Mathematics*, volume III, pages 95–195. John Wiley, New York, 1965.

[9] Daniel Leivant. Logical and mathematical reasoning about imperative programs. In *Conference Record of the Twelfth Annual Symposium on Principles of Programming Languages*, pages 132–140, New York, 1985. ACM.

[10] Daniel Leivant. Matching explicit and modal reasoning about programs: A proof theoretic delineation of dynamic logic. In *Twenty-first Symposium on Logic in Computer Science (LiCS'06)*, pages 157–166, Washington, 2006. IEEE Computer Society Press.

[11] Daniel Leivant. Reasoning in dynamic logic about program termination. In *Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot*, LNCS vol. 4800, pages 394–409. Springer-Verlag, 2007.

[12] Per Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 63–92, Amsterdam, 1971. North-Holland.

[13] Albert Meyer and Joseph Halpern. Axiomatic definition of programming languages: a theoretical assessment. *Journal of the ACM*, 29:555–576, 1982.

[14] Albert Meyer and John Mitchell. Termination assertions for recursive programs: completeness and axiomatic definability. *Information and Control*, 56:112–138, 1983.

[15] D. Prawitz. *Natural Deduction*. Almqvist and Wiksell, Uppsala, 1965.

[16] Ildiko Sain. An elementary proof for some semantic characterizations of nondeterministic Floyd-Hoare logic. *Notre Dame Journal of Formal Logic*, 30:563–573, 1989.

[17] Peter H. Schmitt. Diamond formulas: A fragment of Dynamic Logic with recursive enumerable validity problem. *Information and Computation*, 61:147–158, 1984.