Recursion in Logics of Programs

David Harel †

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

The problem of reasoning about recursive programs is considered. Utilizing a simple analogy between iterative and recursive programs viewed as unfinite unions of finite terms, we carry out an investigation analogous to that carried out recently for iterative programs. The main results are the arithmetical completeness of axiom systems for (1) *context-free dynamic logic* and (2) its extension for dealing with *infinite computations*. Having the power of expression of these logics in mind, these results can be seen to supply (as corollaries) complete proof methods for the various kinds of correctness of recursive programs.

1. Introduction

A majority of the work done to date in reasoning about computer programs seems to be concerned with *regular* programs (e.g. *while* programs or flowcharts etc.). Some of the better known examples are [6,7,8,9,10,19,21,25,28]. When reading this and other relevant literature it becomes apparent that when these programs are replaced by *recursive* programs (i.e *context-free* programs) the problems become much harder. For example, the important paper of de Bakker and Meertens [3] seemed to indicate that Floyd-Hoare-like methods for proving the partial correctness of recursive programs using invariant assertions, required *infinitely many* such assertions. Subsequent work by Greibach [13], Gorelick [12], Apt and Meertens [1], Harel, Pnueli and Stavi [17,18] and Gallier [11], pointed to the fact that this was not quite so. Namely, by allowing the assertion preceding a recursive call to "freeze" the values of all the program variables in order to refer to them upon completion of that call, a natural extension of the Floyd-Hoare technique to recursive programs is possible, using only finitely many assertions. Other work has been done regarding recursive programs: Hitchcock and Park [20] have equated the termination of a recursive program with the well-foundedness of a certain binary relation, and [17] and [29] contain approaches to the problem of axiomatizing the total correctness of (deterministic) recursive programs. Major parts of all of [1,12,17,18,29] are devoted to one or both of the tasks of (1) designing "substitution rules" or "rules of adaptation" for dealing with parameters or local variables, and (2) presenting the methods for mutual recursion. Our point is that these, and other apparent complications, tend to obscure whatever basic concepts and ideas were involved in the essence of the augmentation from regular to context-free reasoning.

This paper is concerned with the problem of clean reasoning about recursive programs. We establish a very simple analogy between iterative and recursive programs and use it to obtain results, similar in spirit to those known for iterative ones, but usually somewhat harder to come by.

In order to be able to put this analogy to good use we choose to carry out our investigation with a simple recursive program construct which we write $\tau^*(f)$. This is the program, specified by the program-term $\tau(X)$, which consists of executing τ with every appearance of the special "place-holder" symbol X standing for a recursive call to τ. Conventional notations for $\tau^*(f)$ include $\mu X\tau(X)$ [5,20], motivated by work on the least fixpoints of functionals. The analogous simple *iterative* construct is α^*, namely the program consisting of carrying out α any ≥ 0 number of times. With *true*? and *false*? standing respectively for the identity (skip) and empty (abort) programs, we define

$$\alpha^0 = \textit{true?} \qquad \tau^0(\textit{false?}) = \textit{false?},$$
$$\alpha^{i+1} = \alpha;\alpha^i \qquad \tau^{i+1}(\textit{false?}) = \tau(\tau^i(\textit{false?})),$$

where $\alpha;\beta$ is the usual *composition* of programs and $\tau(\beta)$ is the *application* of programs, i.e. τ with all free appearances of X replaced by β. The meanings of α^* and $\tau^*(f)$ are binary relations over states, defined respectively (inductively, that is assuming knowledge of the semantics of α and $\tau(\beta)$ for any β) as follows:

$$m(\alpha^*) = \bigcup_{i=0}^{\infty} m(\alpha^i) \qquad m(\tau^*(f)) = \bigcup_{i=0}^{\infty} m(\tau^i(\textit{false?})).$$

This then, illustrates the essence of the analogy: the iterative (recursive) construct is an infinite union of terms consisting of finitary composition (application) of simpler ones, starting with the skip (abort) program.

In most of the recent work on *dynamic logic* and its variants [14,15,16,19,28] emphasis has been placed on working with a simple but powerful iterative programming language, and part of the appeal of this work seems to come from the simplicity which a construct such as α^* brings with it. The work we report upon here makes use of the experience and

† Present address: Mathematical Sciences Department, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.

knowledge attained in [14,15,19] for the regular operator α^*, in order to investigate the problems encountered when reasoning about recursive programs via the simple but powerful context-free analogue, $\tau^*(f)$.

Although we would like to stress that we regard the *approach* developed in this paper, and the potential it might carry for further research, as its most important contribution, we point to the specific results proved:

Context-free dynamic logic (CFDL) is defined and an axiomatization R of it constructed. R is proved to be complete for CFDL relative to arithmetical universes (*arithmetically* complete [14,15]). This result subsumes those of [1,3,11,12,13,17,18] much as the completeness of the axiomatization of DL in [14,15,16] subsumes Cook's [8] completeness result for Hoare's [21] system for partial correctness. As an upshot, it also settles the problem of supplying a complete natural axiomatization of the total correctness of (deterministic) recursive programs. The result also seems to answer a question of de Bakker [2] regarding the weakest precondition [9] of recursive programs. In passing it is shown that the main rule of inference for the partial correctness of recursive programs appearing in [1,11,12,13,18] is simply an instance of Park's [27] fixpoint induction principle.

Next, the question of the *divergence* of a recursive program is considered, prompting us to extend the binary relation semantics of the programs to *computation-tree* semantics. This gives rise to the definition of the special formula $loop_\alpha$ for a program α, being true in a state whenever α, started in that state, can diverge (enter an infinite loop). This concept has been shown to be essential for the definition of the total correctness and weakest precondition of a nondeterministic program [15,19,22] and, besides a similar concept defined in [20] for different purposes, has not been defined yet for recursive programs. We prove, using a result of Winklmann [31], that $loop_\alpha$ is expressible in CFDL. However, the proof results in an extremely undesirable formula, justifying the addition of $loop_\alpha$ to CFDL as a primitive. This addition, we show, gives rise to an arithmetically complete axiomatization of the resulting logic.

2. Context-free Dynamic Logic (CFDL)

Syntax:

We are given sets of *function symbols* and *predicate symbols*, each symbol with a fixed nonnegative *arity*. We assume the inclusion of the special binary predicate symbol "=" (equality) in the latter set. We denote predicate symbols by p, q,... and k-ary function symbols for k>0 by f, g,... Zeroary function symbols are denoted by z,x,y,... and are called *variables*. A *term* is some k-ary function symbol followed by a k-tuple of terms, where we restrict ourselves to terms resulting from applying this formation rule finitely many times only. For a variable x we abbreviate x() to x, thus f(g(x),y) is a term provided f and g are binary and unary respectively. An *atomic formula* is a k-ary predicate symbol followed by a k-tuple of terms. The set of first-order formulae is defined as the closure, under $\vee$, $\neg$ and $\exists x$ (for variable x), of the set of atomic formulae.

We define the set T of *program terms* as follows, using the special "place-holder" symbol X:

(1) For any variable x, term e and first-order formula P, the assignment $x \leftarrow e$, the test P? and the symbol X are all in T,
(2) For all program terms τ_1 and τ_2, $\tau_1;\tau_2$, $\tau_1 \cup \tau_2$ and $\tau_1^*(f)$ are in T. (Remark: f is not a function symbol, it is merely of mnemonic value and is used to signify the difference between α^* and τ^*.)

An occurrence of the symbol X in a term τ_1 is said to be *bound* if it is in a subterm of the form $\tau_2^*(f)$, and *free* otherwise. A term with no free occurrences of any program variable is called *closed*. The $\tau^*(f)$ clause is intended, intuitively, to represent the program consisting of an execution of τ where the free occurrences of the symbol X in τ represent τ *calling* itself recursively. The set CF of *context-free* programs is taken to be the set of closed terms in T. For example, $(y \leftarrow y+1;X \cup y>0?)^*(f)$ is a legal program in CF.

The set of well formule formulae of *Context-free* DL (CFDL-wffs) is defined just as DL in [14,15,16], but using CF instead of the set RG of regular programs:

(1) Any atomic formula is a CFDL-wff,
(2) For any CFDL-wffs P and Q, α in CF and variable x, $\neg P$, $(P \vee Q)$, $\exists x P$ and $\langle\alpha\rangle P$ are CFDL-wffs.

We use $\wedge$, $\supset$, $\equiv$ and $\forall$ as abbreviations in the standard way, and in addition abbreviate $\neg\langle\alpha\rangle\neg P$ to $[\alpha]P$. Note that a first-order formula is a CFDL-wff. Throughout, we denote by $var(\alpha)$ the tuple consisting, in some fixed order, of all variables appearing in α to the left of a $\leftarrow$ symbol

Semantics:

The semantics of CFDL is based on the concept of a state. A *state* J consists of a non empty domain D and a mapping from the sets of function and predicate symbols to the sets of functions and predicates over D, such that to a k-ary function symbol f (resp. predicate symbol p) there corresponds a total k-ary function (resp. predicate) over D denoted by f_J (resp. p_J). In particular, to a variable there corresponds an element of the domain and to a 0-ary predicate symbol (propositional letter) a truth value (*true* or *false*). The standard equality predicate over D is that corresponding to the equality symbol (=). We will sometimes refer to the domain of J as D_J.

The value of a term e = f(e1,...,ek) in a state $\mathcal{J}$ is defined inductively following Tarski [30], by

$$e_{\mathcal{J}} = f_{\mathcal{J}}(e1_{\mathcal{J}},...,ek_{\mathcal{J}}).$$

We now define by simultaneous induction the binary relation over Γ, the set of all states, corresponding to a program α of CF, and those states $\mathcal{J}$ in Γ which satisfy a CFDL-wff P. The relation will be denoted by $m(\alpha)$ and for the latter we write $\mathcal{J}\models P$. $(\mathcal{J},\mathcal{J})$ being an element of $m(\alpha)$ can be thought of as representing the fact that there exists a *computation sequence* (or *path*) of α starting in state $\mathcal{J}$ and terminating in $\mathcal{J}$. Thus, $\mathcal{J}\models[\alpha]P$ will be seen to be making an assertion about *all* terminating computations of α starting in state $\mathcal{J}$; namely the assertion that the final states of these computations satisfy P. Similarly, $\mathcal{J}\models\langle\alpha\rangle P$ asserts the *existence* of a terminating computation of α starting in state $\mathcal{J}$ and ending in a state satisfying P.

Notation: For any function C: D → D', arbitrary element e, and a∈D, we define $[e/a]C$ to be the function with domain D and range D'∪{e} giving the same values at points in D-{a} as C, and such that C(a)=e.

(a) For any variable x and term e,
$m(x\leftarrow e) = \{(\mathcal{J},\mathcal{J}) \mid \mathcal{J}=[e_{\mathcal{J}}/x]\mathcal{J}\}$,

(b) for any program-free CFDL-wff P,
$m(P?) = \{(\mathcal{J},\mathcal{J}) \mid \mathcal{J}\models P\}$,

(c) For any α and β in CF and term τ in T,
$m(\alpha;\beta) = m(\alpha) \circ m(\beta)$, (composition of binary relations)
$m(\alpha\cup\beta) = m(\alpha) \cup m(\beta)$, (union of binary relations)
$m(\tau^*(f)) = \bigcup_{i=0}^{\infty} m(\tau^i(\mathit{false}?))$,

where $\tau^0(\alpha)=\alpha$, $\tau^{i+1}(\alpha)=\tau(\tau^i(\alpha))$ and $\tau(\beta)$ is the CF program obtained by replacing every free occurrence of X in τ by β.

(d) For an atomic formula p(e1,...,ek),
$\mathcal{J}\models p(e1,...,ek)$ whenever $p_{\mathcal{J}}(e1_{\mathcal{J}},...,ek_{\mathcal{J}})$ is true,

(e) For any CFDL-wffs P and Q, α in CF and variable x,
$\mathcal{J}\models\neg P$ iff it is not the case that $\mathcal{J}\models P$,
$\mathcal{J}\models(P\vee Q)$ iff either $\mathcal{J}\models P$ or $\mathcal{J}\models Q$,
$\mathcal{J}\models\exists x P$ iff there exists $d\in D_{\mathcal{J}}$ such that $[d/x]\mathcal{J}\models P$,
$\mathcal{J}\models\langle\alpha\rangle P$ iff there exists $\mathcal{J}\in\Gamma$ such that $(\mathcal{J},\mathcal{J})\in m(\alpha)$ and $\mathcal{J}\models P$.

We will be interested in special subsets of Γ namely simple universes. A *pseudo-universe* U is a set of states all of which have a common domain D. A function symbol f (resp. predicate symbol p) is called *uninterpreted in* U if for every state $\mathcal{J}\in U$ and for every function F (resp. predicate P) over D there exists $\mathcal{J}\in U$ such that $\mathcal{J}$ and $\mathcal{J}$ differ at most in the value of f (resp. p) which in $\mathcal{J}$ is F (resp. P).

A symbol is called *fixed in* U if its value is the same in all states of U. Thus, "=" is fixed in any universe. A *universe* is a pseudo-universe in which every predicate symbol is fixed and in which every function symbol is either fixed or uninterpreted. A universe is called *simple* if the only uninterpreted symbols in it are a designated set of variables. In a simple universe the fixed variables will sometimes be called *constants* following ordinary usage.

In this paper we will primarily be interested in investigating the truth of CFDL-wffs in a given simple universe U. However, one can see that for some $\mathcal{J}\in U$ and some assignment x←e the unique state $\mathcal{J}$ such that $(\mathcal{J},\mathcal{J})\in m(x\leftarrow e)$, i.e. the state $[e_{\mathcal{J}}/x]\mathcal{J}$, might not be in U at all. We outlaw this phenomenon by adopting, from now on, the convention that in the context of a given universe the only programs we consider are those in which the variables assigned to (e.g. x in x←e) and the quantified variables (e.g. x in ∃xP) are uninterpreted. Thus, for $\mathcal{J}\in U$ and for any CFDL-wff P the truth of $\mathcal{J}$ in P can be seen to depend only on states in U. We will often omit the adjective "simple" when no confusion can arise.

An *arithmetical universe* A is a universe in which the domain includes the set of natural numbers, the binary function symbols + and · are fixed and given their standard meanings (addition and multiplication respectively) when applied to the natural numbers in the domain, and 0 and 1 are fixed zeroary-order function symbols interpreted as the natural numbers "zero" and "one" respectively. Furthermore there is a fixed unary predicate symbol *nat* with the interpretation "$\mathit{nat}_{\mathcal{J}}(d)$ is true iff d is a natural number", that is, for every state $\mathcal{J}$, $\{d\in D_{\mathcal{J}} \mid \mathit{nat}_{\mathcal{J}}(d)\}$ is the set of natural numbers. Thus, we are able to distinguish the natural numbers in the domain from the other elements and we do not care, say, what the value of x+y is in state $\mathcal{J}$ when it is not the case that $\mathit{nat}_{\mathcal{J}}(x_{\mathcal{J}})$ holds. An additional property we require of an arithmetical universe is the ability to encode finite sequences of elements into one element. The formal definition of this property is as follows: There exists a total predicate R(x,i,y) over the domain of A, such that for any natural number n it is the case that we have

$$(\forall x_1...x_n)(\exists y)(\forall x\forall i)((\mathit{nat}(i)\wedge n\geq i) \supset (R(x,i,y) \equiv x=x_i)).$$

The intuition is that R(x,i,y) holds iff "x is the i'th component of y", so that any finite sequence $x_1...x_n$ can be encoded as such a y. Note that one particular arithmetical universe is the universe *N* of "pure arithmetic"; that is, the universe in which the domain is precisely the set of natural numbers, and +, ·, 0, = and *nat* (which in this case is identically true), are the only function and predicate symbols. Goedel's β-function serves as the finite sequence encoding function for this universe.

When talking about arithmetical universes we will often want to use n, m,...to stand for variables ranging only over the natural numbers. We do this by adopting the following convention: any L-wff we will use in which we have explicitly mentioned, say, the variable n as a free variable, is assumed to be preceded by "$\mathit{nat}(n)\supset$". Thus, for example, $\mathcal{J}\models(P(n)\supset Q)$ stands for $\mathcal{J}\models(\mathit{nat}(n)\supset(P(n)\supset Q))$, asserting that in state $\mathcal{J}$, $(P(n)\supset Q)$ is true if $n_{\mathcal{J}}$ happens to be a natural number. Furthermore, by convention, $\forall nP(n)$ stands for $\forall n(\mathit{nat}(n)\supset P(n))$, and hence $\exists nP(n)$ abbreviates $\exists n(\mathit{nat}(n)\wedge P(n))$.

It is important to note that any universe U can be extended to an arithmetical universe by augmenting it, if necessary, with the natural numbers and additional apparatus for encoding finite sequences. Thus, reasoning about *any* kind of program, written over any domain, can in principle be carried out with a suitable arithmetical universe.

A CFDL-wff P is U-*valid* (written $\models_U P$) if it is true in all states of the universe U, and is *valid* ($\models P$) if it is U-valid for every universe U.

Example: Consider the program

α: $z \leftarrow x;\ ((z=0?;y \leftarrow 1) \cup (z \neq 0?;z \leftarrow z-1;\ X\ ;z \leftarrow z+1;y \leftarrow y \cdot z))^*(f)$

which is of the form $z \leftarrow x;\tau^*(f)$. The following is $\tau^3(\mathit{false}?)$:

$$((z=0?;y \leftarrow 1) \cup (z \neq 0?;z \leftarrow z-1;$$
$$((z=0?;y \leftarrow 1) \cup (z \neq 0?;z \leftarrow z-1;$$
$$((z=0?;y \leftarrow 1) \cup (z \neq 0?;z \leftarrow z-1;$$
$$\mathit{false}?;$$
$$z \leftarrow z+1;y \leftarrow y \cdot z));$$
$$z \leftarrow z+1;y \leftarrow y \cdot z));$$
$$z \leftarrow z+1;y \leftarrow y \cdot z)).$$

In general it can be seen that the meaning $m(\alpha)$ restricted to the universe $\boldsymbol{N}$ of pure arithmetic is the binary relation $\{(\mathcal{I},\mathcal{J}) |\ \mathcal{J} = \mathcal{I}(x_{\mathcal{J}})!/y]\mathcal{I}\}$, and thus α is a program computing *factorial* over the natural numbers. ∎

One can see that $(\mathcal{I},\mathcal{J}) \in m(\tau^*(f))$ iff there exists a nonnegative integer n such that $(\mathcal{I},\mathcal{J}) \in m(\tau^n(\mathit{false}?))$. In other words the intuition is that "executing" a recursive program $\tau(X)$ which "calls itself" in effect at each appearance of X, is executing, for some n, the program consisting of allowing calls of at most "depth" n.

(We remark that in fact this definition is in perfect agreement with *fixpoint semantics* of recursive programs, as defined, say, in [4] or [5]. Using terminology from these papers our τ's are all *continuous* over the domain of binary relations, and therefore defining the meaning of $\tau^*(f)$ to be the *least* solution of the relational equation $X = \tau(X)$, in the sense of [4] and [20], is, by Kleene's [23] theorem, consistent with our definition of $\tau^*(f)$, which is really $\mu X \tau(X)$.)

In the sequel we will need a fundamental notion to aid in constructing our rules of inference and in conducting our meta-reasoning. We allow in this reasoning a special kind of "program", the *achieve* program P^Z, defined for any first-order formula P and tuple Z consisting of pairwise disjoint variables. The meaning of P^Z is given by the following additional clause to the definition of m:

$$m(P^Z) = \{(\mathcal{I},\mathcal{J}) |\ \mathcal{J} = \mathcal{I}[V/Z]\mathcal{I} \text{ for some tuple V of elements from } D_{\mathcal{I}}, \text{ and } \mathcal{I}[Z_{\mathcal{J}}/Z']\mathcal{I} \models P\}.$$

Thus, P is thought of as having free variables Z and Z', where Z' is a tuple of "primed versions" of the members of Z. For example, (x,y)' is (x',y'). Intuitively then, P^Z is the program which assigns (nondeterministically) to Z any value V such that in state $\mathcal{I}$ P is true of the value of Z in $\mathcal{I}$ and V. Thus, P^Z "achieves" between $\mathcal{I}$ and $\mathcal{J}$ the relation induced by P(Z,Z').

Example: With Z=(x) and P(Z,Z') being $(x'=x \vee x'=f(x))$, we have $m(P^Z) = m(\mathit{true}? \cup x \leftarrow f(x)))$. ∎

Now let CF' = CF ∪ $\{P^Z |$ P and Z as above$\}$. CF'DL is defined exactly as CFDL, but replacing CF with CF'. Our axioms and rules will take advantage of being able, in arithmetical universes, to construct an "achieve program" of the form P^Z to correspond to a given "real" program (i.e. a first-order formula P which "expresses" the relation induced by the program).

3. Preliminary Results

The facts established in this section are instrumental in establishing the main result of Section 4. First we show that the regular programs are embedded in CF. Recall that α^* was defined in [28] such that $m(\alpha^*)=(m(\alpha))^*$, i.e. the reflexive transitive closure of $m(\alpha)$.

Lemma 1: For every $\alpha \in$ CF,
$m(\alpha^*) = m((\mathit{true}? \cup \alpha;X)^*(f)) = m((\mathit{true}? \cup X;\alpha)^*(f))$.

Proof: $m(\alpha^*) = \bigcup_{i=0}^{\infty} m(\alpha^i) = m(\mathit{true}?) \cup m(\alpha) \cup m(\alpha;\alpha) \cup \ldots = m(\mathit{false}?) \cup m(\mathit{true}?) \cup m(\alpha;\mathit{true}?) \cup m(\alpha;\alpha;\mathit{true}?) \cup \ldots = \bigcup_{i=0}^{\infty} m((\mathit{true}? \cup \alpha;X)^i(\mathit{false}?)) = m((\mathit{true}? \cup \alpha;X)^*(f))$. Similarly for the second equality. ∎

Theorem 2: For any arithmetical universe A, the first-order language is A-expressive for CFDL, i.e. for any CFDL-wff P there exists a first-order formula, Q, such that $\models_A (P \equiv Q)$.

Proof: The Theorem is proved similarly to Theorem 3.2 in [15] and the reader is refered to that reference for details. However, here a slightly different treatment for $\tau^*(f)$ is necessary. It can be shown, by the encoding of finite sequences of elements of the domain of A that there exists, for every term $\tau(X)$, a first-order formula $ITR_\tau(n)$ such that for every n $ITR_\tau(n)$ "expresses" $\tau^n(\mathit{false}?)$, in the sense that $m(ITR_\tau(n)^Z) = m(\tau^n(\mathit{false}?))$, where $Z=var(\tau)$. As in Theorem 3.2 of [15], if Q_L is a first-order equivalent of Q then an equivalent of $\langle\tau^*(f)\rangle Q$ is $\exists n \exists Z'(nat(n) \wedge ITR_\tau(n) \wedge (Q_L)^{Z'}_{Z})$. ∎

In the sequel we will write Z=Z' to abbreviate $\wedge_{x \in Z}(x=x')$, and by convention will assume that for programs of the form P^Z, the tuples Z and Z' appear in that order in the paranthesised list of free variables of P. Thus for

example, $P(Z'',Z')$ will abbreviate P with members of Z" replacing free corresponding members of Z. We now show how to express the fact that P^Z is an upper or lower bound on the relation represented by a program α, using notions from CFDL.

Theorem 3: For any universe U and $\alpha \in CF'$, let $m_U(\alpha)$ be $m(\alpha)$ restricted to elements of U. If $Z=var(\alpha)$ then

(1) $\models_U(Z'=Z \supset [\alpha]P(Z',Z))$ iff $m_U(\alpha) \subseteq m_U(P^Z)$,

(2) $\models_U(P(Z,Z') \supset \langle\alpha\rangle Z'=Z)$ iff $m_U(P^Z) \subseteq m_U(\alpha)$.

Proof: (1): Assume $\models_U(Z'=Z \supset [\alpha]P(Z',Z))$ and assume $(\mathcal{I},\mathcal{J}) \in m_U(\alpha)$. We have to show that $\mathcal{J}=[V/Z]\mathcal{I}$ for some tuple V of elements of $D_{\mathcal{J}}$, and that $[Z_{\mathcal{J}}/Z']\mathcal{I} \models P(Z,Z')$. The first is trivial by the fact that $Z=var(\alpha)$. Now, by the definition of $m_U(\alpha)$, and since α does not change Z', if $(\mathcal{I},\mathcal{J}) \in m_U(\alpha)$ then also $(\mathcal{I}',\mathcal{J}') \in m_U(\alpha)$, where $\mathcal{I}'=[Z_{\mathcal{J}}/Z']\mathcal{I}$ and $\mathcal{J}' = [Z_{\mathcal{J}}/Z']\mathcal{J} = [Z_{\mathcal{J}}/Z'][Z_{\mathcal{J}}/Z]\mathcal{I}$. However, by the assumption, since we have constructed $\mathcal{I}'$ such that $\mathcal{I}' \models (Z=Z')$, we must have $\mathcal{J}' \models P(Z',Z)$, or $[Z_{\mathcal{J}}/Z'][Z_{\mathcal{J}}/Z]\mathcal{I} \models P(Z',Z)$, which is the same as saying $[Z_{\mathcal{J}}/Z']\mathcal{I} \models P(Z,Z')$.

Conversely, assume $m_U(\alpha) \subseteq m_U(P^Z)$, and assume that for some $\mathcal{I} \in U$ we have $\mathcal{I} \models (Z'=Z)$, and that $(\mathcal{I},\mathcal{J}) \in m_U(\alpha)$. We must show that $\mathcal{J} \models P(Z',Z)$. By assumption, $(\mathcal{I},\mathcal{J}) \in m_U(P^Z)$, so that $[Z_{\mathcal{J}}/Z']\mathcal{I} \models P(Z,Z')$, which by $\mathcal{I} \models (Z'=Z)$ is equivalent to $[Z_{\mathcal{J}}/Z]\mathcal{I} \models P(Z',Z)$. However, by $(\mathcal{I},\mathcal{J}) \in m_U(\alpha)$ we know that $\mathcal{J}=[Z_{\mathcal{J}}/Z]\mathcal{I}$, so that $\mathcal{J} \models P(Z',Z)$.

(2): Assume $\models_U(P(Z,Z') \supset \langle\alpha\rangle Z'=Z))$, and assume $(\mathcal{I},\mathcal{J}) \in m_U(P^Z)$. We prove $(\mathcal{I},\mathcal{J}) \in \alpha$. By the second assumption, $[Z_{\mathcal{J}}/Z']\mathcal{I} \models P(Z,Z')$, so that by the first we have $([Z_{\mathcal{J}},Z']\mathcal{I}$, $[Z_{\mathcal{J}}/Z][Z_{\mathcal{J}}/Z']\mathcal{I}) \in m_U(\alpha)$. Thus, we can conclude that $(\mathcal{I}$, $[Z_{\mathcal{J}},Z]\mathcal{I}) \in m_U(\alpha)$. Finally, from $\mathcal{J}=[V/Z]\mathcal{I}$ for some V we conclude that $\mathcal{J}=[Z_{\mathcal{J}},Z]\mathcal{I}$, and hence that $(\mathcal{I},\mathcal{J}) \in m_U(\alpha)$.

Conversely, assume $m_U(P^Z) \subseteq m_U(\alpha)$, and that for some $\mathcal{I} \in U$, $\mathcal{I} \models P(Z,Z')$. We show the existence of $\mathcal{J} \in U$ such that $(\mathcal{I},\mathcal{J}) \in m_U(\alpha)$ and $Z'_{\mathcal{J}}=Z_{\mathcal{J}}$. Take $\mathcal{J}$ to be $[Z'_{\mathcal{I}}/Z]\mathcal{I}$. Certainly $Z'_{\mathcal{J}}=Z_{\mathcal{J}}$. Furthermore, by the definition of P^Z, since $[Z'_{\mathcal{I}}/Z']\mathcal{I}$ is simply $\mathcal{I}$ itself, and since we assumed that $\mathcal{I} \models P(Z,Z')$, we conclude that $(\mathcal{I},\mathcal{J}) \in m_U(P^Z)$, and hence $(\mathcal{I},\mathcal{J}) \in m_U(\alpha)$. ∎

It is straightforward to show that our τ's are *monotonic* in the sense that if $m(\alpha) \subseteq m(\beta)$ then $m(\tau(\alpha)) \subseteq m(\tau(\beta))$. We restate Park's fixpoint induction principle:

Lemma 4 (Park [27]): For any universe U, $\alpha \in CF$ and term $\tau(X)$, if $m_U(\tau(\alpha)) \subseteq m_U(\alpha)$ then $m_U(\tau^*(f)) \subseteq m_U(\alpha)$.

Lemma 5: For every universe U, $\alpha_0, \alpha_1, \ldots \in CF'$, and term $\tau(X)$, if $m_U(\alpha_0)=\emptyset$ and if furthermore for all $i \geq 0$ we have $m_U(\alpha_{i+1}) \subseteq m_U(\tau(\alpha_i))$, then for all $i \geq 0$, $m_U(\alpha_i) \subseteq m_U(\tau^*(f))$.

Proof: By induction on i. For i=0 we have $m_U(\alpha_1) \subseteq m_U(\tau(\alpha_0))=m_U(\tau(false?)) \subseteq (\cup^{\infty}_{n=0} m_U(\tau^n(false?)))= m_U(\tau^*(f))$. Assume $m_U(\alpha_i) \subseteq m_U(\tau^*(f))$, so that by monotinicity $m_U(\tau(\alpha_i)) \subseteq m_U(\tau(\tau^*(f)))$. Thus we have $m_U(\alpha_{i+1}) \subseteq m_U(\tau(\alpha_i)) \subseteq m_U(\tau(\tau^*(f)))$. However, one can show by induction on the structure of τ that $m_U(\tau(\cup^{\infty}_{n=0}\tau^n(false?))) = \cup^{\infty}_{n=0} m_U(\tau(\tau^n(false?)))$. (This follows from the continuity of τ over the domain of binary relations; cf. [51].) And so we have $m_U(\alpha_{i+1}) \subseteq \cup^{\infty}_{n=1} m_U(\tau^n(false?))=m_U(\tau^*(f))$. ∎

4. Axiomatization of CFDL

Consider the following axiom system $\boldsymbol{R}$ for CFDL:

Axioms:

(A) All tautologies of propositional calculus.

(B) $[x \leftarrow e]P \equiv P^e_x$, for a first-order formula P.

(C) $[Q?]P \equiv (Q \supset P)$.

(D) $[\alpha;\beta]P \equiv [\alpha][\beta]P$.

(E) $[\alpha \cup \beta]P \equiv ([\alpha]P \wedge [\beta]P)$.

(F) $[P^Z]Q \equiv (\forall Z'')(P(Z,Z'') \supset Q^{Z''}_{Z'})$ for first-order formulae P and Q.

(G) $(P \supset [\tau^*(f)]Q) \supset ((P \wedge R) \supset [\tau^*(f)](Q \wedge R))$ where $var(R) \cap var(\tau)=\emptyset$.

Inference rules:

(H) $$\frac{P \quad , \quad P \supset Q}{Q}$$

(I) $$\frac{P \supset Q}{[\alpha]P \supset [\alpha]Q} \quad \text{and} \quad \frac{P \supset Q}{\exists x P \supset \exists x Q}$$

(J) $$\frac{Z'=Z \supset [\tau(P^Z)]P(Z',Z)}{Z'=Z \supset [\tau^*(f)]P(Z',Z)} \quad \text{where } Z=var(\tau),$$

(K) $$\frac{P(n+1,Z,Z') \supset \langle\tau(P(n)^Z)\rangle Z=Z' \quad , \quad \neg P(0,Z,Z')}{P(n,Z,Z') \supset \langle\tau^*(f)\rangle Z=Z'}$$
for a first-order formula P with $Z=var(\tau)$, $n \notin var(\tau)$.

For any arithmetical universe A denote by $\boldsymbol{R}(A)$ the axiom system obtained by augmenting $\boldsymbol{R}$ with the set of all A-valid

first-order formulae as additional axioms. Provability in $\boldsymbol{R}(A)$ is defined as usual. The intuition behind axiom (G) is that it allows "carrying" R across a program when that program cannot affect the truth of R. The intuition in rule (J) is that if upon "freezing" the values of Z in Z' execution of τ, with the relation P "plugged in" whenever a recursive call was to be performed, results in P holding between the initial values and the current ones, then P holds when the recursive calls are indeed honestly carried out.

We now prove the soundness of $\boldsymbol{R}$.

Lemma 6: For any first-order formulas T and P(Z,Z'), CF'DL-wffs Q, R and S, term $\tau(X)$, the following are valid

(1) $[P^Z]T \equiv (\forall Z')(P(Z,Z') \supset T^{Z'}_{Z})$,

(2) $(S \supset [\tau^*(f)]Q) \supset ((S \wedge R) \supset [\tau^*(f)](Q \wedge R))$, where $var(R) \cap var(\tau) = \emptyset$

Proof: Straightforward from the definitions. ∎

Lemma 7: For any universe U, first-order formula P(Z,Z') and term τ, where $Z = var(\tau)$, if $\models_U (Z'=Z \supset [\tau(P^Z)]P(Z',Z))$ then $\models_U (Z'=Z \supset [\tau^*(f)]P(Z',Z))$.

Proof: By Theorem 3(1) the hypothesis is simply $m_U(\tau(P^Z)) \subseteq m_U(P^Z)$. By Park's principle (Lemma 4) we obtain $m_U(\tau^*(f)) \subseteq m_U(P^Z)$, which, again by Theorem 3(1), is precisely the conclusion. ∎

Lemma 8: For any first-order formula P(n,Z,Z') and term τ, where $n \notin var(\tau)$ and $Z = var(\tau)$, if $\models_A \neg P(0,Z,Z')$ and $\models_A (P(n+1,Z,Z') \supset \langle\tau(P(n)^Z)\rangle Z=Z')$, then $\models_A (P(n,Z,Z') \supset \langle\tau^*(f)\rangle Z=Z')$.

Proof: One can show that $\models_A \neg P(0,Z,Z')$ is in fact equivalent to saying that $m_A(P(0)^Z) = \emptyset$. Furthermore, by Theorem 3(2) the second assumption amounts to asserting that $m_A(P(n+1)^Z) \subseteq m_A(\tau(P(n)^Z))$. By Lemma 5 we conclude that $m_A(P(n)^Z) \subseteq m_A(\tau^*(f))$ for all n. Thus, again by Theorem 3(2), we have the conclusion. ∎

Theorem 9 (A-soundness of $\boldsymbol{R}$): For any CF'DL-wff P,
if $\vdash_{\boldsymbol{R}(A)} P$ then $\models_A P$.

Proof: Follows from the soundness of our axiomatization of regular DL (cf. Thm. 3.6 of [15]) and Lemmas 6,7,8. ∎

We will apply Theorem 11 of [14] (which is Theorem 3.1 of [15]) to prove the arithmetical completeness of $\boldsymbol{R}$, but we are required first to prove the appropriate completeness theorems for formulae with one box or one diamond. These will be established with the aid of:

Lemma 10: The following are derived rules of $\boldsymbol{R}$ where Z and n are as in (J) and (K):

$$(J') \quad \frac{Z'=Z \supset [\tau(P^Z)]P(Z',Z) \quad , \quad R \supset [P^Z]Q}{R \supset [\tau^*(f)]Q}$$

$$(K') \quad \frac{P(n+1,Z,Z') \supset \langle\tau(P(n)^Z)\rangle Z=Z' \quad , \quad \neg P(0,Z,Z') \quad , \quad R \supset \exists n \langle P(n)^Z\rangle Q}{R \supset \langle\tau^*(f)\rangle Q}$$

Proof: (J'): Assume $\vdash_{\boldsymbol{R}(A)} (Z'=Z \supset [\tau(P^Z)]P(Z',Z))$. We apply (J) to obtain $\vdash_{\boldsymbol{R}(A)} (Z'=Z \supset [\tau^*(f)]P(Z',Z))$. Using axiom (G) we get $\vdash_{\boldsymbol{R}(A)} ((Z'=Z \wedge (\forall Z")(P(Z',Z) \supset Q^{Z"}_{Z})) \supset [\tau^*(f)](P(Z',Z) \wedge (\forall Z")(P(Z',Z") \supset Q^{Z"}_{Z})))$, from which we deduce $\vdash_{\boldsymbol{R}(A)} ((\forall Z")(P(Z,Z") \supset Q^{Z'}_{Z}) \supset [\tau^*(f)]Q)$. Thus, by axiom (K) and the second assumption the conclusion follows.
(K'): Similar to (J'). ∎

We now show that rule (J') can indeed always be applied when its conclusion is A-valid.

Lemma 11 (Invariance Lemma for CF'DL): For every term $\tau(X)$ and CF'DL-wffs R and Q, if $\models_A (R \supset [\tau^*(f)]Q)$ then there exists a first-order formula P(Z,Z') with $Z = var(\tau)$, such that
$\models_A (R \supset [P^Z]Q)$ and $\models_A (Z'=Z \supset [\tau(P^Z)]P(Z',Z))$.

Proof: Implied by the way Theorem 2 is proved is the fact that there exists a first order formula of arithmetic P(Z,Z') which "expresses" the program $\tau^*(f)$ in the sense that $m_A(P^Z) = m_A(\tau^*(f))$. Certainly then, by the assumption, we have $\models_A (R \supset [P^Z]Q)$. Also, as noted in the proof of Lemma 5, $m_A(\tau(\tau^*(f))) = m_A(\tau^*(f))$, and so we have $m_A(\tau(P^Z)) \subseteq m_A(P^Z)$, which by Theorem 3(1) is $\models_A (Z'=Z \supset [\tau(P^Z)]P(Z',Z))$. ∎

Theorem 12 (Box-completeness Theorem for CF'DL): For every $\alpha \in CF'$ and first-order formulas R and Q, if $\models_A (R \supset [\alpha]Q)$ then $\vdash_{\boldsymbol{R}(A)} (R \supset [\alpha]Q)$.

Proof: Straightforward proof by induction on the structure of α. The connectives $\leftarrow$, ?, ; and $\cup$ are treated precisely as in [14,15]. The case $\tau^*(f)$ is treated using derived rule (J') and Lemma 11. ∎

Similarly to Lemma 11, rule (K') can be applied when needed:

Lemma 13 (Convergence Lemma for CF'DL): For every term $\tau(X)$ and CF'DL-wffs R and Q, if $\models_A (R \supset \langle\tau^*(f)\rangle Q)$ then there exists a first-order formula P(n,Z,Z') such that $\models_A (P(n+1,Z,Z') \supset \langle\tau(P(n)^Z)\rangle Z=Z')$, $\models_A \neg P(0,Z,Z')$, and $\models_A (R \supset \exists n \langle P(n)^Z\rangle Q)$.

Proof: Again, by the method used in the proof of Theorem 2, there exists a first-order formula $P(n,Z,Z')$ representing $\tau^n(false?)$ in the sense that for every n we have $m_A(P(n)^Z)=m_A(\tau^n(false?))$. It is easy to see that all three A-validities hold for P. ∎

Theorem 14 (Diamond-completeness Theorem for CF'DL): For every $\alpha\in CF'$ and first-order formulas R and Q, if $\models_A(R\supset\langle\alpha\rangle Q)$ then $\vdash_{R(A)}(R\supset\langle\alpha\rangle Q)$.

Proof: Precisely as Theorem 12, but using Lemma 13 and rule (K') instead of Lemma 11 and rule (J'). ∎

We conclude that for CF'DL-wffs, A-validity and provability in $R(A)$ are equivalent concepts:

Theorem 15 (Arithmetical Soundness and Completeness for CF'DL): For every arithmetical universe A and CF'DL-wff P,

$$\models_A P \quad \text{iff} \quad \vdash_{R(A)} P.$$

Proof: One direction is Theorem 9, and the other follows from the general Theorem 11 of [14] (which is Thm. 3.1 of [15]) and the present Theorems 2, 12 and 14. ∎

The results in this section indicate that, as far as CFDL is concerned, reasoning about "pure" recursion is analogous to (albeit more complicated than) that of reasoning about "pure" iteration. Here we are using the integers to count how "deep" we are in the recursion (using $P(n)^Z$ in rule (K)), whereas for α^* we counted how "far" we are in the iteration.

It is interesting to note that the proof method for formulae of the form $R\supset[\alpha]Q$ which is incorporated into R boils down to Floyd's [10] *inductive assertion* method and to Morris and Wegbreit's [26] *subgoal induction* method respectively, when regular programs are translated into recursive ones via the two methods appearing in Lemma 1. Thus the duality holding between these two methods, which was described in length in [26], shows up concisely as stemming from the two dual ways of viewing α^*. The reader familiar with [26] can convince himself of this fact quite easily by deriving formulae (3.1-3.3) and (3.4-3.6) of [26] from our rule (J) by using the two equalities of Lemma 1.

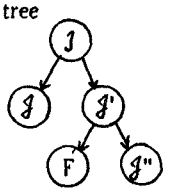
In Section 4.4 of [15] we extend this system to the general *mutual recursion* operator $\mu_i X_1 \ldots X_n(\tau_1,\ldots,\tau_n)$. Appendices of [15] contain examples of proofs in R and R^+ of Section 6.

5. Divergence of Recursive programs

In this section we define the operational notion of the *diverging* of a recursive program, i.e. it entering an infinite loop. This is done by introducing, for any state $\mathcal{I}$, the *$\mathcal{I}$-computation tree* of a program α, denoted by $ct(\alpha,\mathcal{I})$. We show that computation trees are in fact an extension of the semantics of CF. The trees however, in addition to the input-output information, contain information regarding the presence or absence of divergences. The concept of diverging has been shown in [15,19,22] to be essential for describing the total correctness of nondeterministic programs, and hence the importance of investigating it for recursive programs as well as iterative ones. The main result in this section, based on a result of K. Winklmann, is the fact that diverging is expressible in CFDL.

Each node of $ct(\alpha,\mathcal{I})$ will be labeled with a state in Γ or with the special symbol F, and will be of outdegree at most 2. The root is labeled with $\mathcal{I}$ and nodes labeled with F will always be leaves. The intuition is that a path from the root represents a legal computation of α starting in state $\mathcal{I}$. Accordingly, a leaf represents a termination state if it is labeled with a state in Γ, or reaching a false test if it is labeled with F. Any node with descendants represents an intermediate state of α. If a node has two descendants then there is, so to speak, a choice as to how to "continue execution".

A node will be represented by a pair (t,I), where t is a finite string over {0,1} describing the location of the node in the tree by 0 denoting "go left" and 1 "go right", and I (the label of the node) is either a state in Γ or the symbol F. Thus, for example, the tree



is represented as $\{(\lambda,\mathcal{I}),(0,\mathcal{J}),(1,\mathcal{J}'),(10,F),(11,\mathcal{J}'')\}$. As can be seen, λ, the empty string, marks the root of the tree. By convention, a single descendant is marked as "going left", i.e. by 0.

Formally, for any $\mathcal{I}\in\Gamma$ and $\alpha\in RC$, $ct(\alpha,\mathcal{I})$ is defined, by induction on the structure of α, to be a subset of $\{0,1\}^* \times (\Gamma\cup\{F\})$ as follows, where we use I to range over $(\Gamma\cup\{F\})$, and s, t,... to range over $\{0,1\}^*$:

(1) $ct(x\leftarrow e,\mathcal{I}) = \{(\lambda,\mathcal{I}),(0,[e_{\mathcal{I}}/x]\mathcal{I})\}$,

(2) $ct(P?,\mathcal{I}) = \begin{cases}\{(\lambda,\mathcal{I})\} & \text{if } \mathcal{I}\models P\\ \{(\lambda,F)\} & \text{if } \mathcal{I}\models\neg P,\end{cases}$

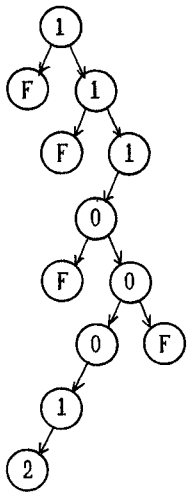
(3) $ct(\alpha\cup\beta,\mathcal{I}) = \{(\lambda,\mathcal{I})\}\cup\{(0t,I)\mid(t,I)\in ct(\alpha,\mathcal{I})\}\cup\{(1t,I)\mid(t,I)\in ct(\beta,\mathcal{I})\}$,

(4) Let $E=\{(t,\mathcal{J})\in ct(\alpha,\mathcal{I})\mid \mathcal{J}\in\Gamma\wedge(\forall b\in\{0,1\})(\forall I\in(\Gamma\cup\{F\}))((tb,I)\notin ct(\alpha,\mathcal{I}))\}$, and let $C=ct(\alpha,\mathcal{I})-E$. Then $ct(\alpha;\beta,\mathcal{I}) = C\cup\{(ts,I)\mid(\exists\mathcal{J})((t,\mathcal{J})\in E\wedge(s,I)\in ct(\beta,\mathcal{J}))\}$,

(5) $ct(\tau^*(f),\mathcal{I}) = ct(false?\cup\tau(\tau^*(f)),\mathcal{I})$

Informally, to construct $ct(\tau^*(f),\mathcal{I})$ one starts constructing $ct(false? \cup \tau(X))$, and whenever, so to speak, "$ct(X,\mathcal{J})$" was to be attached to a node labeled $\mathcal{J}$, $ct(\tau^*(f),\mathcal{J})$ is attached instead. Of course, this process might lead to an infinite tree. The additional union with *false*? is introduced so that the process of calling recursively would itself "cost" an edge in the tree, so that e.g. the program $(X)^*(f)$ will diverge (see below).

Example: Let $\mathcal{I}$ be some state in $\mathbf{N}$ for which $y_{\mathcal{I}}=0$, and in the diagram we let i stand for $[i/y]\mathcal{I}$. Take α to be the program
$((y=0?;y\leftarrow y+1) \cup (y\neq 0?;y\leftarrow y-1;X;y\leftarrow y+1))^*(f)$.
Then $ct(\alpha,[1/y]\mathcal{I})$ is



∎

We omit the proof of the following fact, which serves to show that computation trees are consistent with the binary relation semantics defined above:

Lemma 16: For every $\alpha\in CF$, $(\mathcal{I},\mathcal{J})\in m(\alpha)$ iff $ct(\alpha,\mathcal{I})$ has a leaf labeled $\mathcal{J}$.

Define now, for any $\alpha\in CF$ a Boolean constant $loop_\alpha$ by:

$$\mathcal{I}\models loop_\alpha \quad \text{iff} \quad ct(\alpha,\mathcal{I}) \text{ is infinite,}$$

Note that, $ct(\alpha,\mathcal{I})$ being of finite outdegree, we can apply Koenig's lemma (see [24]) to conclude that in fact $\mathcal{I}\models loop_\alpha$ iff there exists an *infinite path* from the root; i.e. there is an infinite sequence of nodes in $ct(\alpha,\mathcal{I})$, of the form

$$(\lambda,\mathcal{I}), (b_1,\mathcal{I}_1), (b_1b_2,\mathcal{I}_2), \ldots, (b_1\ldots b_i,\mathcal{I}_i), \ldots$$

We would like to supply a syntactic characterization of $loop_{\tau^*(f)}$. Recall that in [19] it was proved that a divergence in the iterative α^* is due either to a local divergence, i.e. a divergence in some reachable execution of α, or to a global one, i.e. being able to execute α's for ever. The former possibility was captured by $\langle\alpha^*\rangle loop_\alpha$ which can be written $\exists n loop_\alpha n$, and the latter is $\forall n\langle\alpha^n\rangle true$. So we can write

$$\models (loop_{\alpha^*} \equiv (\exists n loop_\alpha n \vee \forall n\langle\alpha^n\rangle true)).$$

Characterizing $loop_{\tau^*(f)}$ is similar; here a local divergence is a divergence "inside" some application of a reachable τ, and can be expressed by $\exists n loop_{\tau^n(false?)}$. Global diverging, on the other hand, is more subtle. Here we want to express the possibility of being able to "apply τ for ever", which amounts to being able to "proceed infinitely deep into the recursion".

In order to do this we apply, in the sequel, the following mild restriction on the universes we consider. The domain is to have at least two distinct elements and the set of symbols is to include two fixed variables having these two elements as values. We will therefore use the symbols a and b freely as two fixed-valued variables with distinct values. Now, for any term τ, define the term τ' which, intuitively, allows "skipping" tests, recursive calls to τ, and other recursive constructs, but forces any such skip to be recorded in a new "flag" variable x. given $\tau(X)$, let $x,y\notin var(\tau)$ be two variables, and let $\tau'(X)$ be $\tau(X)$ with every (free in the case of X) appearance of a subterm α of one of the forms X, P? or $\tau''^*(f)$ replaced by $(\alpha \cup x\leftarrow b)$. Also define

$$\sigma: \quad (y\neq a? \cup (y=a?;x=a?;y\leftarrow b)).$$

For any $n\geq 0$ denote the program $x\leftarrow a;y\leftarrow a;\tau'^n(\sigma)$ by τ_n. We can now present our characterization of $loop_{\tau^*(f)}$:

Theorem 17: For any $\alpha\in CF$,
$$\models (loop_{\tau^*(f)} \equiv (\exists n loop_{\tau^n(false?)} \vee \forall n\langle\tau_n\rangle y=b)).$$

Proof: Assume we have $\mathcal{I}\models\exists n loop_{\tau^n(false?)}$. It is quite easy to see that $ct(\tau^*(f),\mathcal{I})$ has at least as many nodes as $ct(\tau^n(false?),\mathcal{I})$, and hence we also have $\mathcal{I}\models loop_{\tau^*(f)}$.

For the rest of the proof we will be needing some additional notation. For any $i\geq 0$ and $\mathcal{I}\in\Gamma$ we would like to define the set $S(i,\tau,\mathcal{I})$ consisting of those states which occur immediately before an application of τ at "depth i". Define

$$S(0,\tau,\mathcal{I}) = \{\mathcal{I}\},$$
$$S(i+1,\tau,\mathcal{I}) = \cup_{\mathcal{J}\in V} S(i,\tau,\mathcal{J}),$$

where V is the set of states $\mathcal{J}$ such that the process of constructing $ct(\tau(x\leftarrow x;\tau^*(f)),\mathcal{I})$ for $x\notin var(\tau)$ requires constructing $ct(x\leftarrow x,\mathcal{J})$. In other words, V is the set of states which execution of $\tau^*(f)$ can reach just prior to calling τ recursively at level 1. Certainly if for some i we have $\mathcal{J}\in S(i,\tau,\mathcal{I})$, then $\mathcal{J}$ labels some node in $ct(\tau^*(f),\mathcal{I})$, and furthermore the path from the root of $ct(\tau^*(f),\mathcal{I})$ to that node is of length at least 1. (Note that this would not be the case if we were to define $ct(\tau^*(f),\mathcal{I})$ to be $ct(\tau(\tau^*(f)),\mathcal{I})$.)

Assume now that $\mathcal{I}\models\forall n\langle\tau_n\rangle y=b$. We show that for any $i\geq 0$ we have $S(i,\tau,\mathcal{I})\neq\emptyset$, and thus $ct(\tau^*(f),\mathcal{I})$ has paths of

arbitrary length and is therefore, by Koenig's Lemma, infinite. (Note that the assumption $\exists^{\infty}n\langle\tau_n\rangle y=b$ is sufficient, where $\exists^{\infty}n$ reads "there exist infinitely many n", so that $\forall n$ can be replaced by $\exists^{\infty}n$ in the statement of the Theorem.) Indeed, for any such i, by assumption, we have $\mathcal{I}\models\langle\tau_i\rangle y=b$, or $\mathcal{I}\models\langle x\leftarrow a;y\leftarrow a;\tau'^{i}(\sigma)\rangle y=b$, so that there exists a finite path p in $ct(\tau_i,\mathcal{I})$, starting from the root, which terminates in a node labeled by a state in which the value of y is b. The labels of the successive nodes of p can be denoted by

$$(\mathcal{I}\ ,\ [a/x]\mathcal{I}\ ,\ [a/y][a/x]\mathcal{I}\ ,\ \mathcal{J}0\ ,\ \ldots\ ,\ \mathcal{J}k)$$

where $y_{\mathcal{J}k}=b$. Let i be the least integer such that $y_{\mathcal{J}i}=b$. By the construction of $\tau'(\sigma)$ it is evident that in order for y to have changed value from a to b, it must be the case that the value of x was a all along. More precisely, for all $j\geq j'\geq 0$ we have $x_{\mathcal{J}j'}=a$, so that tests P? and subprograms of the form $\tau''^{*}(f)$ were indeed "carried out" and not avoided by executing $x\leftarrow b$ instead. In other words, the initial segment of the path p ending in $\mathcal{J}j$ can be thought of as being a simulation, in $\tau'^{i}(\sigma)$, of a path from the root to the *false*? in $ct(\tau^{i}(false?),\mathcal{I})$. Consequently, we have $\mathcal{J}j\in S(i,\tau,\mathcal{I})$. This completes the proof of one direction of the theorem.

Conversely, assume now that $\mathcal{I}\models loop_{\tau^{*}(f)}$ holds and that for all $n\geq 0$ we have $\mathcal{I}\models\neg loop_{\tau^{n}(false?)}$. Consider the infinite sequence s of successive labels of the nodes of an infinite path from the root in $ct(\tau^{*}(f),\mathcal{I})$. It is easy to see that by the second hypothesis, there must exist a subsequence of s, say $(\mathcal{J}0,\mathcal{J}1,\ldots)$, such that for every i we have $\mathcal{J}i\in S(i,\tau,\mathcal{I})$ and such that $\mathcal{J}i$ corresponds to the first time in s that "depth i" of recursion was reached. We show that $\mathcal{I}\models\langle\tau_i\rangle y=b$ holds for every i by giving an algorithm for executing τ_i in such a way as to terminate in a state in which the value of y is b. Given i, simulate the path corresponding to the initial segment of the sequence s ending in $\mathcal{J}i$, i.e. assign $x\leftarrow a$ and $y\leftarrow a$, and then proceed in $\tau'^{i}(\sigma)$ exactly as s proceeded in $\tau^{*}(f)$, executing tests and recursive constructs and *not* the $x\leftarrow b$ parts. By the definition of $\mathcal{J}i$, reaching $\mathcal{J}i$ in s corresponds to reaching σ for the first time in $\tau'^{i}(\sigma)$. Thus, we have reached σ with $y_{\mathcal{J}i}=a$ and $x_{\mathcal{J}i}=a$ and therefore y is assigned b. Execution in $\tau'^{i}(\sigma)$ is then to be continued by choosing the $x\leftarrow b$ parts instead of tests, appearances of X and recursive constructs. Certainly this execution will terminate (no tests to fail; no recursive constructs or recursive calls to diverge). Moreover, by the construction of σ any subsequent arrival at σ will not change the value of y, and since $y\notin var(\tau)$, this value is not changed by any other part of the rest of the execution. Thus, y=b upon termination. ∎

An obvious question arising now is whether $loop_{\alpha}$ is expressible in CFDL, i.e. whether for any $\alpha\in CF$ there exists a CFDL-wff P_{α} such that $\models(P_{\alpha}\equiv loop_{\alpha})$. We have to be able to deal with both disjuncts of Theorem 17.

We state the following two results and prove the second. The proof of the first, due to K. Winklmann, is omitted here.

Theorem 18 (Winklmann [31]): For every term $\tau(X)$, $\alpha\in CF$ and first-order formula P there exists a CFDL-wff Q such that
$$\models(Q\equiv\exists^{\infty}n\langle\tau^{n}(\alpha)\rangle P).$$

Theorem 19: For every term $\tau(X)$ there exists a CFDL-wff Q such that $\models(Q\equiv\exists n loop_{\tau^{n}(false?)})$.

And hence from Theorems 17, 18 and 19, and the remark in the proof of Thm. 17, which justifies replacing $\forall n$ in its statement by $\exists^{\infty}n$, we obtain:

Corollary 20: $loop_{\alpha}$ is expressible in CFDL.

Proof of Theorem 19: Consider the set $\Sigma_{\mathcal{I}}^{\tau}=\cup_{i=0}^{\infty}S(i,\tau,\mathcal{I})$ which, intuitively, is the set of states labeling those nodes in $ct(\tau^{*}(f),\mathcal{I})$ corresponding to points just prior to a recursive call to τ. Assume we have defined, for any CFDL-wff Q and term $\tau(X)$, a formula $along(\tau,Q)$ such that

$$\mathcal{I}\models along(\tau,Q)\quad\text{iff}\quad(\exists\mathcal{J}\in\Sigma_{\mathcal{I}}^{\tau})(\mathcal{J}\models Q),$$

i.e. $\mathcal{I}\models along(\tau,Q)$ holds iff Q is true immediately prior to some reachable recursive call to τ in an execution of $\tau^{*}(f)$ starting in state $\mathcal{I}$. Assume also that we have defined, for every program $\alpha\in CF$ and term $\tau(X)$, a formula $lp_{\tau,\alpha}$ such that, intuitively, $\mathcal{I}\models lp_{\tau,\alpha}$ holds iff there is a divergence in $ct(\tau(\alpha),\mathcal{I})$ which is due to the τ part and not to the α part (i.e. the divergence came from some recursive construct appearing in $\tau(X)$). It is quite clear that $\mathcal{I}\models\exists n loop_{\tau^{n}(false?)}$ holds iff at some state $\mathcal{J}$ in the execution of $\tau^{*}(f)$ just prior to a recursive call to τ, it is the case that there is a divergence in $ct(\tau(\tau^{*}(f)),\mathcal{J})$ which is due to the first τ and not to the inner $\tau^{*}(f)$. In other words $\mathcal{I}\models along(\tau,\ lp_{\tau,\tau^{*}(f)})$. Now we proceed to define these concepts, and then observe that they give rise to CFDL-wffs.

For any $\alpha,\beta,\beta'\in CF$ and terms $\tau1(X)$ and $\tau2(X)$ define

$$\begin{aligned}
&lp_{X,\alpha} =_{df} false,\\
&lp_{\beta,\alpha} =_{df} loop_{\beta},\\
&lp_{\beta;X,\alpha} =_{df} loop_{\beta},\\
&lp_{X;\beta,\alpha} =_{df} \langle\alpha\rangle loop_{\beta},\\
&lp_{\beta;X;\beta',\alpha} =_{df} (loop_{\beta}\vee\langle\beta\rangle lp_{X;\beta',\alpha}),\\
&lp_{\tau1\cup\tau2,\alpha} =_{df} (lp_{\tau1,\alpha}\vee lp_{\tau2,\alpha}).
\end{aligned}$$

Now for defining $along(\tau,Q)$ we use tricks similar to those used in constructing τ' and σ for Theorem 17. Given $\tau(X)$, let $x,y\notin var(\tau)$ be two variables, let $Z=var(\tau)$ and let Z' be a tuple of disjoint primed versions of the variables in $var(\tau)$.

Define $\tau''(X)$ to be $\tau(X)$ with every appearance of a subprogram α of the form P? or $\tau'''^*(f)$ replaced by $(\alpha \cup x \leftarrow b)$, and every appearance of the program variable X replaced by

$$((x=a?;y=a?;y\leftarrow b;Z'\leftarrow Z) \cup x\leftarrow b \cup X)$$

where $Z'\leftarrow Z$ abbreviates the composition of the assignments $z'\leftarrow z$ for all $z\in Z$. Now, define $along(\tau,Q)$ to be

$$\langle x\leftarrow a;y\leftarrow a;\tau''^*(f)\rangle(y=b \wedge \langle Z\leftarrow Z'\rangle Q).$$

The intuition is that in $x\leftarrow a;y\leftarrow a;\tau''^*(f)$ one has the option of, whenever X is reached, storing the current values of the variables Z in Z', as long as the computation until that point has been an honest simulation of a computation in $\tau^*(f)$. Once such a store has been carried out it cannot be carried out again because of the y=a? guard. Furthermore, as in the proof of Theorem 17, execution can always choose to "surface" quickly to the end of $\tau''^*(f)$ by executing $x\leftarrow b$ whenever possible. Then, when the execution finally terminates, we assert that Q is true for the values of Z which we stored in Z' just before the recursive call. It should now be clear that

$$\models(\exists n\, loop_{\tau^n(false?)} \equiv along(\tau, lp_{\tau,\tau^*(f)})).$$

The reason the theorem now follows is that, when using induction on the structure of τ and assuming the theorem holds for the subterms of τ, we can deduce that our inductive hypothesis is in fact that $loop_\alpha$ is expressible in CFDL for α which appears in τ. This follows from Theorems 17 and 18. Thus, the definition of lp, which uses $loop_\alpha$ for such α, gives rise to a CFDL-wff. ∎

6. Augmented CFDL

In this section we augment CFDL with the ability to reason about divergences directly without having to go through the translation of $loop_\alpha$ into its equivalent CFDL-wff. The resulting language will be called CFDL$^+$. The reason for doing so, even though by the previous section CFDL and CFDL$^+$ are equivalent in expressive power, is rooted in the fact that the CFDL equivalent of $loop_\alpha$ supplied above (including the part coming from the omitted construction in the proof of Thm. 18) has the unpleasant property of being strongly dependent on the structure of α and on the variables appearing in α. Calling that equivalent P_α, one can see that $P_{\alpha'}$ cannot be obtained from P_α by substituting α' for α throughout. Consequently, proving a formula with an appearance of $loop_\alpha$ will inevitably involve carrying out the transformation of $loop_\alpha$ to P_α, and then reasoning in CFDL. The point is that the intuition one might have about $loop_\alpha$ is, in a strong sense, lost in the process.

CFDL$^+$ is defined to be CFDL $\cup$ $\{loop_\alpha \mid \alpha\in CF\}$, with the semantics of the CFDL part being as in CFDL and the semantics of $loop_\alpha$ as defined in Section 5.

Our axiomatization here too will be of an extension CF'DL$^+$ which is defined as CFDL$^+$ but with the programs coming from the set CF'. As in Section 4, we will be using the fact that in an arithmetical universe A there exists, for any $\alpha\in CF$, a first-order formula P such that P^Z expresses α. The problem that arises is that of defining $loop_{(P^Z)}$. We would like it to be the case that for any P^Z, $\models\neg loop_{(P^Z)}$ holds. However, for a given $\mathcal{I}\in\Gamma$ it is possible that the set $\mathcal{I}(P^Z) = \{\mathcal{J} \mid (\mathcal{I},\mathcal{J})\in m(P^Z)\}$ is infinite. One solution to this problem is to define $ct(\alpha,\mathcal{I})$ to be a tree of possibly *infinite* outdegree, with the location of the node given by a list of natural numbers (as opposed to a list, or string, of 0's and 1's); for P^Z the tree would be defined (roughly) as

$$ct(P^Z,\mathcal{I}) = \{(\lambda,\mathcal{I})\} \cup \{(i,\mathcal{J}_i) \mid (\mathcal{I},\mathcal{J}_i)\in m(P^Z)\}.$$

Then, we would define $\mathcal{I}\models loop_\alpha$ to hold iff $ct(\alpha,\mathcal{I})$ has an infinite path (which in this case is not necessarily equivalent to $ct(\alpha,\mathcal{I})$ being infinite).

Another, equivalent method is to associate with any $\alpha\in CF'$ and $\mathcal{I}\in\Gamma$ a *set* of computation trees $CT(\alpha,\mathcal{I})$. For P^Z we would define

$$CT(P^Z,\mathcal{I}) = \{\{(\lambda,\mathcal{I}),(0,\mathcal{J})\} \mid (\mathcal{I},\mathcal{J})\in m(P^Z)\}.$$

The rest of the definition is carried out analogously to the definition of $ct(\alpha,\mathcal{I})$ above. For example, $CT(\alpha;\beta,\mathcal{I})$ is the set of trees obtained by following the construction of $ct(\alpha;\beta,\mathcal{I})$ for every tree in $CT(\alpha,\mathcal{I})$, attaching any tree in $CT(\beta,\mathcal{J})$ to a node labeled $\mathcal{J}$ whenever $ct(\beta,\mathcal{J})$ was to be attached to that node in constructing $ct(\alpha;\beta,\mathcal{I})$.

Example: Let α: $x\leftarrow x+1$, P: $x<x'$ and $Z=(x)$. For any $\mathcal{I}\in\mathbf{N}$ such that $x_{\mathcal{I}}=0$ we have:

$$CT(\alpha,\mathcal{I}) = \{\{(\lambda,\mathcal{I}),(0,[1/x]\mathcal{I})\}\},$$

$$CT(P^Z,[1/x]\mathcal{I}) = \{\{(\lambda,[1/x]\mathcal{I}),(0,\mathcal{J})\} \mid x_{\mathcal{J}}>1\},$$

and thus

$$CT(\alpha;P^Z,\mathcal{I}) = \{\{(\lambda,\mathcal{I}),(0,[1/x]\mathcal{I}),(00,\mathcal{J})\} \mid x_{\mathcal{J}}>1\}.$$ ∎

Now define $\mathcal{I}\models loop_\alpha$ to hold iff there is an infinite tree in $CT(\alpha,\mathcal{I})$. We remark that either way $loop_\alpha$ is uniquely defined for $\alpha\in CF'$, and that for $\alpha\in CF$, $CT(\alpha,\mathcal{I}) = \{ct(\alpha,\mathcal{I})\}$.

Now consider the axiom system $\mathbf{R}^+$ for CFDL$^+$ defined as $\mathbf{R}$ of Section 4 augmented with the following axioms and rules: (In the following, P and Q are first-order, R is a CF'DL$^+$-wff, $\tau(X)$ is a term, x and y are variables $x,y\notin var(\tau)$, $Z=var(\tau)$, V is the tuple of variables obtained by augmenting Z with x and y, and σ, τ' and τ'' are as defined in the proofs of Theorems 17 and 19 respectively.)

Axioms:

(L) $loop_{x\leftarrow e} \equiv false$,

(M) $loop_{P?} \equiv false$,

(N) $loop_{\alpha;\beta} \equiv (loop_\alpha \vee \langle\alpha\rangle loop_\beta)$,

(O) $loop_{\alpha\cup\beta} \equiv (loop_\alpha \vee loop_\beta)$,

(P) $loop_{(P^Z)} \equiv false$.

Rules of Inference:

(Q)

$$\frac{\begin{array}{l} R \supset (\langle x\leftarrow a;y\leftarrow a;\tau'^*(f)\rangle(y=b \wedge \langle Z\leftarrow Z'\rangle loop_{\tau(Q^Z)}) \vee \\ \qquad\qquad \forall n\langle x\leftarrow a;y\leftarrow a;P(n)^V\rangle y=b) , \\ P(0,V,V') \supset \langle\sigma\rangle V=V' \quad , \quad Q(Z,Z') \supset \langle\tau'^*(f)\rangle Z=Z' , \\ P(n+1,V,V') \supset \langle\tau'(P(n)^V)\rangle V=V' \end{array}}{R \supset loop_{\tau'^*(f)}}$$

(R)

$$\frac{\begin{array}{l} R \supset ([x\leftarrow a;y\leftarrow a;\tau'^*(f)](y\neq b \vee [Z\leftarrow Z']\neg loop_{\tau(Q^Z)}) \wedge \\ \qquad\qquad \exists n[x\leftarrow a;y\leftarrow a;P(n)^V]y\neq b) , \\ V=V' \supset [\sigma]P(0,V',V) \quad , \quad Z=Z' \supset [\tau'^*(f)]Q(Z',Z) , \\ V'=V \supset [\tau'(P(n)^V)]P(n+1,V',V) \end{array}}{R \supset \neg loop_{\tau'^*(f)}}$$

Let A be any arithmetical universe. $R^+(A)$ is defined analogously to $R(A)$. Provability in $R^+(A)$ is defined as usual.

Theorem 21 (A-soundness of R^+): For any CF'DL-wff P, if $\vdash_{R^+(A)} P$ then $\models_A P$.

Proof: The proof of the A-validity of (L)-(P) is left to the reader. We show that rule (Q) is sound, noting that the soundness of the dual rule (R) follows immediately.

Consider rule (Q). We argue that the A-validity of the first premise of this rule, under the assumption that the other three are A-valid, asserts that

$$\models_A (R \supset (\exists n loop_{\tau'^n(false?)} \vee \forall n\langle\tau_n\rangle y=b),$$

which, by Theorem 17, implies that $\models_A (R \supset loop_{\tau'^*(f)})$. (Recall that τ_n is an abbreviation of $(x\leftarrow a;y\leftarrow a;\tau'^n(\sigma))$.) And indeed, by Theorem 3 in Section 3 the premises, other than the first, assert, respectively, $m_A(P(0)^V)\subseteq m_A(\sigma)$, $m_A(Q^Z)\subseteq m_A(\tau'^*(f))$, and $\forall n(m_A(P(n+1)^V)\subseteq m_A(\tau'(P(n)^V)))$. One can then show, by induction on n using monotonicity, that $\forall n(m_A(P(n)^V)\subseteq m_A(\tau'^n(\sigma)))$. Consequently, since Q^Z is "smaller" as a relation than $\tau'^*(f)$ but is divergence-free, one can see that $loop_{\tau(Q^Z)}$ implies $lp_{\tau,\tau'^*(f)}$, and hence also that $along(\tau, loop_{\tau(Q^Z)})$ implies $along(\tau, lp_{\tau,\tau'^*(f)})$. However, by the proof of Theorem 19 the latter is $\exists n loop_{\tau'^n(false?)}$. Moreover, since for any n, $P(n)^V$ is "smaller" than $\tau'^n(\sigma)$, one can see that $\forall n\langle x\leftarrow a;y\leftarrow a;P(n)^V\rangle y=b$ implies $\forall n\langle\tau_n\rangle y=b$. ∎

In order to prove the arithmetical completeness of R^+ we need to show that A-valid CF'DL$^+$-wffs of one of the forms $R\supset loop_\alpha$ or $R\supset\neg loop_\alpha$, where R is first-order, are provable in $R^+(A)$. Then the general theorem (11 in [14], 3.1 in [15]) is used to obtain the final result. More details can be found in [15].

Theorem 22: For every $\alpha\in$CF' and first-order formula R, if $\vdash_A (R\supset loop_\alpha)$ then $\vdash_{R^+(A)} (R\supset loop_\alpha)$.

Proof: By induction on the structure of α. The only nontrivial case is when α is of the form $\tau'^*(f)$ for some term τ. Assuming $\vdash_A (R\supset loop_\alpha)$, we show the existence of first-order formulas Q and P(n) such that the premises of rule (Q) are A-valid. Since these premises involve only CF'DL-wffs and the formula $loop_{\tau(Q^Z)}$, in which the program is of complexity lower than $\tau'^*(f)$, the result will follow. Indeed, by Theorem 2 we can take Q and P(n) to be first-order formulae involving, respectively, only variables in Z and V, and such that $\models_A (Q \equiv \tau'^*(f))$ and for all n $\models_A (P(n) \equiv \tau'^n(\sigma))$. All the premises are easily seen to be A-valid for this choice. ∎

Similarly we have:

Theorem 23: For every $\alpha\in$CF' and first-order formula R, if $\vdash_A (R\supset\neg loop_\alpha)$ then $\vdash_{R^+(A)} (R\supset\neg loop_\alpha)$.

And thus, as remarked, we conclude:

Theorem 24 (Arithmetical Soundness and Completeness for CF'DL$^+$): For every CF'DL$^+$-wff P,

$$\models_A P \quad \text{iff} \quad \vdash_{R^+(A)} P.$$

Acknowledgments

We wish to thank A. Pnueli for suggesting a first version of rule (K), and N. Dershowitz, A. Shamir and V.R. Pratt for helpful discussions. The research was supported by the Yad-Avi Foundation in Israel and by NSF Grant no. MCS76-18461.

References

[1] Apt, K.R. and L.G.L.T. Meertens. Completeness with Finite Systems of Intermediate Assertions for Recursive Program Schemes. IW 84/77, Math. Cent. Amsterdam. Sept. 1977.

[2] deBakker, J.W. Recursive Programs as Predicate Transformers. Proc. IFIP conf. on Formal Specifications of Programming Constructs. St. Andrews, Canada. Aug. 1977.

[3] de Bakker, J.W. and L.G.L.T. Meertens. On the Completeness of the Inductive Assertion Method. J. of Computer and System Sciences, 11, 323-357. 1975.

[4] de Bakker, J.W. and W.P. deRoever. A Calculus for Recursive Program Schemes. in *Automata, Languages and Programming* (ed. Nivat), 167-196. North Holland. 1972.

[5] de Bakker, J.W., and D. Scott. An outline of a theory of programs. Unpublished manuscript, 1969.

[6] Banachowski, L., A. Kreczmar, C. Mirkowska, H. Rasiowa and A. Salwicki. An Introduction to Algorithmic Logic; Metamathematical Investigations in the Theory of Programs. In Mazurkiewitcz and Pawlak (editors) Math. Found. of Comp. Sc. Banach Center Publications. Warsaw. 1977.

[7] Burstall, R.M. Program Proving as Hand Simulation with a Little Induction. IFIP 1974, Stockholm.

[8] Cook, S.A. Soundness and Completeness of an Axiom System for Program Verification. SIAM J. Comp. Vol. 7, no. 1. Feb. 1978. (A revision of: Axiomatic and Interpretive Semantics for an Algol Fragment. TR-79. Dept. of Computer Science, U. of Toronto. 1975.)

[9] Dijkstra, E. W. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. Comm. of the ACM. vol 18, no.8. 1975

[10] Floyd, R.W. Assigning Meaning to Programs. In J.T. Schwartz (ed.) *Mathematical Aspects of Computer Science.* Proc. Symp. in Applied Math. 19. Providence, R.I. American Math. Soc. 19-32. 1967.

[11] Gallier, J. Semantics and Correctness of Nondeterministic Flowcharts with Recursive Procedures. In 5th *Automata, Languages and Programming.* Springer-Verlag. July 1978.

[12] Gorelick, G.A. A Complete Axiomatic System for Proving Assertions about Recursive and Nonrecursive Programs. TR-75. Dept. of Computer Science, U. of Toronto. 1975.

[13] Greibach, S.A. *Theory of Program Structures.* Lecture Notes in Computer Science, 36. Springer-Verlag. 1975.

[14] Harel, D. Arithmetical Completeness in Logics of Programs. In 5th *Automata, Languages and Programming.* Springer-Verlag. July 1978.

[15] Harel, D. Logics of Programs: Axiomatics and Descriptive Power. Ph.D. Thesis. MIT. Available as MIT/ LCS/ TR-200. May 1978.

[16] Harel, D., A.R. Meyer and V.R. Pratt. Computability and Completeness in Logics of Programs. Proc. 9th Ann. ACM Symp. on Theory of Computing, Boulder, Col., May 1977.

[17] Harel, D., A. Pnueli and J. Stavi. Completeness Issues for Inductive Assertions and Hoare's Method. Tech. Rep., Dept. of Applied Math. Tel-Aviv U. Israel. Aug. 1976.

[18] Harel, D., A. Pnueli and J. Stavi. A Complete Axiomatic System for Proving Deductions about Recursive Programs. Proc. 9th Ann. ACM Symp. on Theory of Computing, Boulder, Col., May 1977.

[19] Harel, D. and V.R. Pratt. Nondeterminism in Logics of Programs. Proc. 5th ACM Symp. on Principles of Programming Languages. Tucson, Ariz. Jan. 1978.

[20] Hitchcock, P. and D. Park. Induction Rules and Termination Proofs. In *Automata, Languages and Programming* (ed. Nivat, M.), IRIA. North-Holland, 1973.

[21] Hoare, C.A.R. An Axiomatic Basis for Computer Programming. Comm. of the ACM, vol. 12, 576-580, 1969.

[22] Hoare, C.A.R. Some Properties of Predicate Transformers. JACM, vol.25, no.3. July 1978.

[23] Kleene, S.C. *Introduction to Metamathematics.* D. Van Nostrand. 1952.

[24] Koenig, D. *Theorie der endlichen und unendlichen Graphen.* Leipzig. 1936. Reprinted by Chelsea, New York. 1950.

[25] Manna, Z. and R. Waldinger. Is "Sometime" Sometimes Better than "Always"? Intermittent Assertions in Proving Program Correctness. Proc. 2nd Int. Conf. on Software Engineering, Oct. 1976.

[26] Morris, J.H. Jr. and B. Wegbreit. Subgoal Induction. Comm. of the ACM. vol. 20. no. 4. April 1977.

[27] Park, D. Fixpoint Induction and Proofs of Program Properties. In *Machine Intelligence 5.* Edinburgh University Press. 1969.

[28] Pratt, V.R. Semantical Considerations on Floyd-Hoare Logic. Proc. 17th IEEE Symp. on Foundations of Computer Science. 109-121. Oct. 1976.

[29] Sokolowski, S. Total Correctness for Procedures. Manuscript. Univ. of Gdansk, Poland. 1977.

[30] Tarski, A. The semantic conception of truth and the foundations of semantics. Philos. and Phenom. Res, *4*, 341-376. 1944.

[31] Winklmann, K. private communication.