LABORATORY FOR
COMPUTER SCIENCE
(formerly Project MAC)

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-98

NONDETERMINISM IN LOGICS OF PROGRAMS

DAVID HAREL
VAUGHAN R. PRATT

FEBRUARY 1978

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

MIT/LCS/TM-98

NONDETERMINISM IN LOGIS OF PROGRAMS

David Harel
Vaughan R. Pratt

February 1979

Nondeterminism in Logics of Programs

by

David Harel and Vaughan R. Pratt

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Mass. 02139, 11/10/77

Abstract.

We investigate the principles underlying reasoning about nondeterministic programs, and present a logic to support this kind of reasoning. Our logic, an extension of dynamic logic ([22] and [12]), subsumes most existing first-order logics of nondeterministic programs, including that developed by Dijkstra based on the concept of weakest precondition. A significant feature is the strict separation between the two kinds of nonterminating computations: infinite computations and failures. The logic has a Tarskian truth-value semantics, an essential prerequisite to establishing completeness of axiomatizations of the logic. We give an axiomatization for flowchart (regular) programs that is complete relative to arithmetic in the sense of Cook. Having a satisfactory tool at hand, we turn to the clarification of the concept of the total correctness of nondeterministic programs, providing in passing, a critical evaluation of the widely used "predicate transformer" approach to the definition of programming constructs, initiated by Dijkstra [5]. Our axiom system supplies a complete axiomatization of *wp*.

1. Introduction

1.1. The Interest in Nondeterminism.

Nondeterministic programs have attracted considerable attention lately. This interest could be attributed to a concern for generality: anything we have to say about nondeterministic programs covers deterministic programs as a special case. However there are also deeper reasons for this interest:

First, nondeterministic programs have been proposed [21] as a model of parallel processes. Such parallelism arises in timeshared computers, where nondeterminism expresses the apparent capriciousness of the scheduler. It also arises in the management

Key words: Arithmetical completeness
Divergence
Dynamic logic
Failure
Nondeterminism
Total correctness
Weakest preconditions

of external physical devices, where the nondeterminism captures the unpredictable behavior of physical devices.

Second, nondeterminism is gaining credence as a component of a programming style that imposes the fewest constraints on the processor executing the program. For example a certain program may run correctly provided that initially x is even. If the programmer requires the processor to set x to an even number of the programmer's choosing, the processor may be unduly constrained. On a byte oriented machine where integers are represented as four-byte quantities, setting x to a particular number requires four operations, but if the programmer has merely requested setting it to an arbitrary even number the processor can satisfy the request with one operation, by setting the low-order byte to, say, zero.

Third, nondeterminism supplies one methodology for interfacing two procedures that, though written independently, are intended to cooperate on solving a single problem. The approach is to make one procedure an "intelligent" interpreter for the other. Woods' Augmented Transition Networks [27] supply an instance of the style. The user of this system writes a grammar for a specific natural language which amounts to a nondeterministic program to be run on Woods' interpreter, which though ignorant of the details of specific languages nevertheless contributes much domain-independent parsing knowledge to the problem of making choices left unspecified by the user's program. This technique is in wide use in other areas of Artificial Intelligence, and supplies a way of viewing such AI programming languages as QA-3/QA-4/STRIPS [8], PLANNER [14], etc.

Fourth, from a strictly mathematical viewpoint, there is something dissatisfying about taking such constructs as *if then else* and *while do* as primitive constructs. *If then else* involves the two concepts of *testing* and *choosing*, and *while do* involves the two concepts of *testing* and *iterating*. A more basic approach is to develop these concepts separately. However, in isolating the concept of testing from the concepts of choosing and iterating, we have removed the parts of the *if then else* and *while do* constructs responsible for their determinism.

Fifth, from a practical point of view, when reasoning about deterministic programs it can sometimes be convenient to make what amounts to claims about nondeterministic programs. When we argue that "*if* $x>0$ *then* $x \leftarrow x-1$ *else* $x \leftarrow x+1$" cannot affect y, a part of our argument might be that, whether we execute $x \leftarrow x-1$ or $x \leftarrow x+1$, y will not change. The fact that the whole program is deterministic played no role in this argument, which amounts to the observation that the nondeterministic program $(x \leftarrow x-1 \cup x \leftarrow x+1)$ cannot change y. ($\alpha \cup \beta$ is a program calling for execution of either program α or program β, the choice being made arbitrarily, i.e. nondeterministically.) By the same token the observation that "*while* $x<0$ *do* $x \leftarrow x+2$" leaves the parity of x unchanged depends principally on the fact that executing $x \leftarrow x+2$ arbitrarily often, i.e. executing $(x \leftarrow x+2)^*$, leaves the parity of x unchanged. (α^* is a program calling for a number of executions of program α, the choice of number being made nondeterministically.) This illustrates the appropriateness of applying nondeterministic reasoning to deterministic programs.

1.2. The Interest in Abstract Programs.

Concretely a program is "just its listing," unless we consider its intensional aspects such as its *raison d'etre*, its proof of correctness, or the actual cost of writing it. At any event we shall consider in this paper that its listing supplies everything we know about the program. To facilitate reasoning about a program we shall often find it convenient to *discard* information about that program. The amount of information discarded depends critically on the nature of the reasoning. When the reasoning involves only the so-called input-output behavior of a program, as it does in discussing partial correctness, the appropriate degree of abstraction treats programs as functions on states, or binary relations in the case of nondeterministic programs. However this degree of abstraction is inappropriate when one wants to distinguish between different kinds of nontermination such as diverging versus failing, or when one wants to deal with running time, or space utilization, or the program's interaction with its environment as it runs, or any other aspect of a program not covered by simple initial-state-final-state relationships.

In this paper we shall find it convenient to talk about three kinds of programs, namely concrete programs (listings, or elements of a word-algebra), computation trees, and binary relations on states.

The interest in computation trees is that they exhibit in a natural way just those details relevant to the main problems we address in this paper, namely how to talk about the behavior of nondeterministic programs taking into account pathological behavior such as diverging and failing. A computation tree is a tree whose vertices are states. Each path of the tree represents a possible state trajectory or computation sequence for the program. The root of the tree identifies the starting state of the program.

The interest in binary relations may be attributed to the fact that much reasoning about programs centers on their "external" behavior, the question of which state (or states, in the presence of nondeterminism) the program will ultimately drive the processor into from a given starting state. Such reasoning can often be confined solely to the external behavior of the program and its constituent subprograms, in which case it is convenient to take as the objects under discussion not the programs themselves but merely their behaviors. The appropriate abstract object that associates with each initial state of the world a final state, is a *function* from states to states. When nondeterminism is possible the appropriate choice of object is a *binary relation* on states.

The three levels of abstraction, concrete programs, computation trees, and binary relations, form a hierarchy of levels of increasing abstraction, or equivalently decreasing information. If one were to embed our treatment in a more algebraic framework than we shall do in this paper one would treat the concrete programs as an initial algebra of a category of abstractions, with a chain of arrows (homomorphisms) from the initial algebra to computation trees and thence to binary relations. In fact we shall explicitly exhibit these homomorphisms, but we shall not explicitly adopt a category-theoretic approach in so doing.

1.3. Upper and lower bounds.

Much discussion about programs takes the form of bounds on their behavior. For example we may claim that if and when program α terminates, $x=3$. This is an *upper bound* on the behavior of α in that α may not terminate in states not satisfying $x=3$. It *forbids* transitions having a final state not satisfying $x=3$, but says nothing about the *existence* of transitions. Conversely we may claim that it is always possible for "*while* $x>0$ *do* $x \leftarrow x-1$" to terminate with $x=0$. This is a *lower bound*; it promises the existence of transitions with final state satisfying $x=0$, one such transition for every possible starting state (because of the "always"). However it says nothing in itself forbidding the possibility of other transitions, though in this instance the knowledge that the program is deterministic allows us to *infer* (as an upper bound) the absence of any other transitions. More generally, asserting that a program is deterministic is by itself an upper bound. Asserting that it is total (there always exists a halting computation) amounts to a lower bound.

A substantial difference between our approach to bounds and that implicit in Dijkstra's weakest precondition (*wp*) operator [5] is that we shall at all times keep the upper and lower bounds separate. It will become evident when we come to prove our completeness results that such a separation is essential to the success of an approach such as ours to getting completeness results in this area. In contrast Dijkstra lets the single *wp* operator impose both upper bounds (partial correctness) and lower bounds (proper termination) simultaneously, and we do not see how to deal with the combination in any way that is not equivalent to the decomposition made explicit in our approach.

1.4. Contents.

Elsewhere (see [22] and [12]) we describe a language for reasoning about bounds on abstract programs, which we have called *dynamic logic*; a language sufficiently general that it encompasses the expressive power of most existing first-order languages proposed for this purpose, yet so simple in its conception that it would appear to be suitable as a standard tool supplying convenient terminology for defining the concepts and constructs arising in other languages.

In Section 2 we first recall the basic concepts of dynamic logic (DL) as given in [22] and [12]. We then describe the $\mathcal{I}$-computation-tree and the $\mathcal{I}$-computation sequences of a program α in state $\mathcal{I}$, emphasizing the importance of the notions of *diverging* and *failing*, corresponding respectively to executing an infinite computation, and reaching a false test.

Having the trees of Section 2 in mind, Section 3 (which is the main section of the paper) deals with the extension of DL to DL$^+$, by adding a *divergence state* to the universe U of states, and adding to the $[\alpha]$ and $\langle\alpha\rangle$ modalities of DL, a $[\alpha]^+$ modality with its dual $\langle\alpha\rangle^+$, constructed for facilitating reasoning about the presence and absence of divergences. Various important properties of the new system are proved. The axiom system P which was proved relatively complete for DL in [12], is augmented with two rules, and the resulting P^+ is proved complete for DL$^+$. Thus one can now say and prove e.g. $\langle\alpha\rangle$*true* $\wedge$ $[\alpha]^+P$ of a program α and formula P, meaning "α can terminate, and whenever it does P holds; moreover, there is no way of entering an infinite loop".

Section 4 contains a clarification of the concept of "total correctness" when applied to nondeterministic programs, advancing the argument that this becomes an ill-defined concept unless a strict method of *executing* the programs is adopted. We then carry out a critical investigation of Dijkstra's notion of the *weakest precondition* (*wp*), which is considered to be a basic tool for proving the correctness of nondeterministic programs. The notion of *wp* has been described in [5] and [6] in three different ways, none of which is constructive, and none of which is completely consistent with the others. Observations in this direction have been made in [1], [23] and [15], but we still feel that a mist covers this widely used notion, which seems to have been introduced to the program-proving community with the same strong motivation, but with the same lack of underlying semantics as was Hoare's partial correctness notion P{α}Q. We hope to have achieved a clarification of this concept. Finally, in Section 5 we refer to important work by others related to the topic of our paper.

2. *Dynamic Logic, Computation Trees, Diverging and Failing.*

First we recall the basics of regular DL, described in greater detail in [12]. We assume we are given some universe of states U, the elements of which we denote by $\mathcal{I},\mathcal{J}$ etc. First order formulae are assigned truth values in the states of U by the standard methods, writing $\mathcal{J}\models$P when P evaluates to *true* in $\mathcal{I}$. Furthermore, every regular program α over assignments (simple, in this paper) and tests, is regarded as a binary relation over U in the following manner:

$x\leftarrow E = \{(\mathcal{I},\mathcal{J})|$ every symbol has the same value in $\mathcal{I}$ and $\mathcal{J}$ except x which in $\mathcal{J}$ has the value that the expression E has in $\mathcal{I}\}$,

$P? = \{(\mathcal{I},\mathcal{I})|\ \mathcal{I}\models P\}$,

$\alpha;\beta$, $\alpha\cup\beta$ and α^* are the composition, union and reflexive transitive closure of their components respectively. We will write $\mathcal{I}\alpha\mathcal{J}$ for $(\mathcal{I},\mathcal{J})\in\alpha$. A new formation rule is added to those of predicate calculus by admitting $\langle\alpha\rangle P$ (read "diamond-α P") as a formula for any program α and formula P, with the semantics

$$\mathcal{I}\models\langle\alpha\rangle P \quad \text{iff} \quad \exists\mathcal{J}(\mathcal{I}\alpha\mathcal{J} \wedge \mathcal{J}\models P), \quad \text{or equivalently} \quad \bigvee_{\mathcal{I}\alpha\mathcal{J}} \mathcal{J}\models P.$$

Thus for the dual $\neg\langle\alpha\rangle\neg P$ denoted by $[\alpha]P$ ("box-α P"),

$$\mathcal{I}\models[\alpha]P \quad \text{iff} \quad \forall\mathcal{J}(\mathcal{I}\alpha\mathcal{J} \supset \mathcal{J}\models P), \quad \text{or equivalently} \quad \bigwedge_{\mathcal{I}\alpha\mathcal{J}} \mathcal{J}\models P.$$

With this language many of the properties of programs which relate initial and final states, can be stated and proved, like $\models R\supset[\alpha]Q$ (partial correctness), $\models R\supset\langle\alpha\rangle Q$ (total correctness if α is deterministic) as well as more sophisticated statements such as the valid

$$[(\alpha;\alpha)^*]P \wedge [\alpha;(\alpha;\alpha)^*]\neg P \equiv P \wedge [\alpha^*](P\supset[\alpha]\neg P \wedge \neg P\supset[\alpha]P).$$

Two approaches to the axiomatization of DL can be pursued, namely constructing infinitary axiom systems with the goal of achieving (absolute) completeness (this can be done for DL as work we are doing jointly with A.R. Meyer shows), and a variation of Cook's method, namely constructing finitary axiom systems which are complete relative to arithmetic; i.e. taking as axioms all the valid formulae of first order arithmetic. We would like to reserve for the latter the term *Arithmetical Completeness.* We reproduce here a variant of the axiom system P first appearing in [22], which was proved to be arithmetically complete in [12].

(A) All valid sentences of first order arithmetic,
(B) All tautologies of Propositional Calculus,
(C) $[x\leftarrow E]P \equiv P_x^E$ where P is modality-free,
(D) $[P?]Q \equiv P\supset Q$,
(E) $[\alpha\cup\beta]P \equiv [\alpha]P \wedge [\beta]P$,
(F) $[\alpha;\beta]P \equiv [\alpha][\beta]P$,

(G) $$\frac{P \supset Q}{[\alpha]P \supset [\alpha]Q}$$

(H) Invariance $$\frac{P \supset [\alpha]P}{P \supset [\alpha^*]P}$$

(I) Convergence $$\frac{P(n+1) \supset \langle\alpha\rangle P(n)}{P(n) \supset \langle\alpha^*\rangle P(0)}$$

where $P(t)$ for some arithmetical term t stands for P^t_x, x does not appear in α, and n does not appear in $P(x)$ or α.

We make a side remark here and note that rule (H) can be replaced by the equivalent "induction" axiom $(P \wedge [\alpha^*](P \supset [\alpha]P)) \supset [\alpha^*]P$, and that we have changed the rule of necessitation so as to make possible the elimination of the axiom

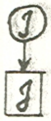
$$[\alpha](P \supset Q) \supset ([\alpha]P \supset [\alpha]Q)$$

of [22] and [12]. Neither change falsifies our completeness result of [12].

Were our programs restricted to be deterministic (say, by replacing $\cup$ and $*$ by some deterministic conditional and iteration programming constructs), DL would suffice. In this case $\langle\alpha\rangle P$ states that "everything will be ok"; the program (via its one and only possible path of execution) will terminate satisfying P. $[\alpha]P$ states that if α terminates then P will be satisfied, whereas capturing the fact that "something goes wrong" can be done with $[\alpha]$*false*, etc. Hence, theories of total correctness of deterministic programs are quite easy to construct and understand; they are based in most cases on one basic construct corresponding to $\langle\alpha\rangle P$ (*sometimes...at...* [19] and [4], P[S]Q [3], [P{S}Q] [26], <P|S|Q> [18], *som* [16] etc.). In [10] we describe a deterministic version of DL, and survey a large number of known methods for reasoning about deterministic programs, arguing that the underlying principles in this case are few and relatively simple.

For the more complex case of nondeterministic programs we will formalize the notion of "executing" a program α in a given state $\mathcal{I} \in U$, by first defining the $\mathcal{I}$*-computation tree* of α, $t(\alpha,\mathcal{I})$. Given $\mathcal{I}$ and α, $t(\alpha,\mathcal{I})$ is a possibly infinite tree, each node of which can have only finitely many descendants, and is labeled with either a state (element of U) or the symbol f together with an indication of whether the node is a "halt" node (which we denote with a square; a circle indicates a non-halt node).

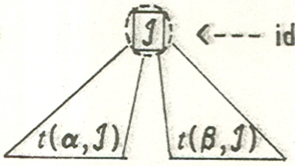
- $t(x \leftarrow E,\mathcal{I})$ is



where $\mathcal{J}$ is the unique state such that $(\mathcal{I},\mathcal{J}) \in x \leftarrow E$,

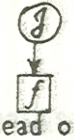
- $t(P?,\mathcal{I})$ is $\boxed{\mathcal{I}}$ if $\mathcal{I} \models P$, and $\textcircled{\mathcal{I}}$ if $\mathcal{I} \models \neg P$,

- $t(\alpha \cup \beta,\mathcal{I})$ is



<--- identification of roots (root is square iff one of component roots is)

- $t(\alpha;\beta,\mathcal{I})$ is the tree resulting from the attachment, for any state $\mathcal{J}$, of $t(\beta,\mathcal{J})$ to every halt node of $t(\alpha,\mathcal{I})$ which is labeled with the state $\mathcal{J}$, except for the case where $t(\beta,\mathcal{J})$ is a single non-halt node and the node to which it is being attached has no descendants, in which case the tree



is used instead of $t(\beta,\mathcal{J})$. The halt nodes of the resulting tree are taken to be just the halt nodes of β (i.e. square nodes of $t(\alpha,\mathcal{I})$ are "rounded" unless they are also square nodes of $t(\beta,\mathcal{I})$).

- $t(\alpha^*,\mathcal{I})$ is the (possibly infinite) tree defined recursively as $t((\mathit{true}? \cup \alpha';\alpha^*),\mathcal{I})$ where α' is α with its root "rounded" (made non-halting).

The reader may verify that computation trees have finite out-degree at every node. In proving this it is helpful to note that if a tree has a leaf that is not a halt node then that leaf is the root. (This would not be the case if we did not "round" the root of α in $t(\alpha^*,\mathcal{I})$.) From the point of view of our definition we are regarding α^* as the "finitely-wide / infinitely-deep" program $I \cup \alpha;(I \cup \alpha;(I \cup \alpha;(...)))$, and not as the "infinitely-wide/ finitely-deep" program $I \cup \alpha \cup \alpha;\alpha \cup \alpha;\alpha;\alpha \cup ...$ (where I abbreviates *true*?).

The reason for not associating an f-node with a false test directly, and instead introducing f-nodes only via ";", is so that failure to satisfy a test does not count as failure if an immediate alternative is provided (necessarily via union). This permits us to retain the definition of "*if* P *then* α *else* β" as $P?;\alpha \cup \neg P?;\beta$. In view of the convenience of reasoning about tests and union independently, it seems to us well worth while to arrange the slippery notion of "failure" so that it does not make this definition unusable later on.

The set of paths of $t(\alpha,\mathcal{I})$ from the root $\mathcal{I}$ will be called the set of *computation sequences* of α and $\mathcal{I}$, $c(\alpha,\mathcal{I})$. We will call the infinite elements of $c(\alpha,\mathcal{I})$ *divergences*, and the finite elements of $c(\alpha,\mathcal{I})$ terminating in nodes labeled f, *failures*. A divergence corresponds to the presence of a possible infinite computation of α, while a failure corresponds to coming across a test evaluating to *false* and having no immediate alternatives to pursue that do not entail backtracking.

These concepts supply the setting and motivation for the following sections.

3. Dynamic Logic$^+$.

In this section we define an extended DL which we will denote by DL$^+$. It will have the ability to express the absence of divergences in $c(\alpha,\mathcal{I})$, by employing the modality $[\alpha]^+$ (and its dual $\langle\alpha\rangle^+$), taking $[\alpha]^+P$ to mean that there are no divergences and that every possible final state satisfies P. In Section 4 it will also become evident that DL$^+$ is powerful enough to express the absence (and hence the presence) of failures too. Thus, a wide range of properties of nondeterministic programs can be expressed in DL$^+$, including, as we will see in Section 4, all the different versions of the notion of total correctness. The central theorems of this section are an inductive characterization of the fact that α^* can (cannot) diverge, and the arithmetical completeness of an axiom system for DL$^+$, which provides for the first time a complete formal proof method for formulae including assertions about infinite loops and failures.

A remark seems to be in place before we proceed. We are about to add a *divergence state* to the universe U, which for lack of a better symbol we denote by $\bot$. This state should not be confused with the similar "undefined state" of say [1], [23] or [17], the difference being that the $\bot$ symbol in these papers stands for a divergence *and* failure state. The pros and cons of this different approach will be analyzed, and arguments for adopting ours will be presented, in Section 4.

Define therefore, for a given universe U, $U^+ =_{df} U\cup\{\bot\}$. Truth in $\bot$ is given by defining $\bot\not\models P$ for every formula P (i.e. the set $\{P|\bot\models P\}$ is empty). Validity in U$^+$ is, however, defined to be validity in U, to avoid losing such familiar theorems as $P\supset P$. New denotations for programs are obtained in DL$^+$ by adding $(\bot,\bot)$ to every test and assignment. The definitions of union and composition remain unchanged. However, we take α^* to be the reflexive transitive closure of α *together with* transitions $(\mathcal{I}_0,\bot)$ when there exist states $\mathcal{I}_1,\mathcal{I}_2,\ldots$ such that $\forall i\geq 0(\mathcal{I}_i\alpha\mathcal{I}_{i+1})$.

(An alternative definition of the binary relation on U$^+$ represented by α is that it is the set of all pairs $(\mathcal{I},\mathcal{J})$ such that $\mathcal{J}$ labels some halt node of $t(\alpha,\mathcal{I})$ not labelled f, together with $(\mathcal{I},\bot)$ whenever $t(\alpha,\mathcal{I})$ is infinite, or equivalently has an infinite path (equivalence follows by Koenig's Lemma and the finite outdegree of the nodes of the tree). This approach simplifies the definition by making the description of computation trees do all the work.)

Now let us reexamine the definitions of $\langle\alpha\rangle$P and $[\alpha]$P. The fact that for $\mathcal{I}\models[\alpha]$P we could write $\forall\mathcal{J}(\mathcal{I}\alpha\mathcal{J} \supset \mathcal{J}\models$P$)$ although $[\alpha]$ was defined as $\neg\langle\alpha\rangle\neg$, depends on the fact that $\mathcal{J}\not\models\neg$P is the same as $\mathcal{J}\models$P for all $\mathcal{J}$. We note that $\bot\not\models$P and $\bot\not\models\neg$P, and consequently $\mathcal{I}\models\neg\langle\alpha\rangle\neg$P is no longer the same as $\forall\mathcal{J}(\mathcal{I}\alpha\mathcal{J} \supset \mathcal{J}\models$P$)$ for all states $\mathcal{I}$. Rather, we now define

$$\mathcal{I}\models[\alpha]^{+}\mathrm{P} \quad \text{iff} \quad \forall\mathcal{J}(\mathcal{I}\alpha\mathcal{J} \supset \mathcal{J}\models\mathrm{P}), \quad \text{or} \quad \bigwedge_{\mathcal{I}\alpha\mathcal{J}} \mathcal{J}\models\mathrm{P},$$

$$\mathcal{I}\models[\alpha]\mathrm{P} \quad \text{iff} \quad \forall\mathcal{J}(\mathcal{I}\alpha\mathcal{J} \supset \mathcal{J}\not\models\neg\mathrm{P}), \quad \text{or} \quad \bigwedge_{\mathcal{I}\alpha\mathcal{J}} \mathcal{J}\not\models\neg\mathrm{P},$$

$$\mathcal{I}\models\langle\alpha\rangle\mathrm{P} \quad \text{iff} \quad \exists\mathcal{J}(\mathcal{I}\alpha\mathcal{J} \wedge \mathcal{J}\models\mathrm{P}), \quad \text{or} \quad \bigvee_{\mathcal{I}\alpha\mathcal{J}} \mathcal{J}\models\mathrm{P},$$

$$\mathcal{I}\models\langle\alpha\rangle^{+}\mathrm{P} \quad \text{iff} \quad \exists\mathcal{J}(\mathcal{I}\alpha\mathcal{J} \wedge \mathcal{J}\not\models\neg\mathrm{P}), \quad \text{or} \quad \bigvee_{\mathcal{I}\alpha\mathcal{J}} \mathcal{J}\not\models\neg\mathrm{P},$$

and we clearly have $[\,]=\neg\langle\,\rangle\neg$ and $[\,]^{+}=\neg\langle\,\rangle^{+}\neg$. (The new modalities are read "box-plus α", and "diamond-plus α".) Inspection shows that these four concepts assert about $c(\alpha,\mathcal{I})$ that (respectively)

there are no divergences and every final state satisfies P,
every final state satisfies P,
there exists a final state satisfying P, and
there exists either a divergence, or a final state satisfying P.

Note that $\mathcal{I}\models[\alpha]^{+}$*true* states the absence of divergences in $c(\alpha,\mathcal{I})$, and $\mathcal{I}\models\langle\alpha\rangle^{+}$*false* the existence of (at least) one.

We now gather some of the basic properties of our new modalities:

Lemma 1. For all programs α and β, DL^{+}-formula P, assignment x←E and test Q?, the following are valid:

(a) $[\alpha]^{+}\mathrm{P} \equiv [\alpha]\mathrm{P} \wedge [\alpha]^{+}\mathit{true}$,
(b) $\langle\alpha\rangle^{+}\mathrm{P} \equiv \langle\alpha\rangle\mathrm{P} \vee \langle\alpha\rangle^{+}\mathit{false}$,
(c) $[\mathrm{x}\leftarrow\mathrm{E}]^{+}\mathit{true}$,
(d) $[\mathrm{Q}?]^{+}\mathit{true}$,
(e) $[\alpha;\beta]^{+}\mathrm{P} \equiv [\alpha]^{+}[\beta]^{+}\mathrm{P} \equiv [\alpha]^{+}\mathit{true} \wedge [\alpha][\beta]^{+}\mathit{true} \wedge [\alpha][\beta]\mathrm{P}$,
(f) $\langle\alpha;\beta\rangle^{+}\mathrm{P} \equiv \langle\alpha\rangle^{+}\langle\beta\rangle^{+}\mathrm{P} \equiv \langle\alpha\rangle^{+}\mathit{false} \vee \langle\alpha\rangle\langle\beta\rangle^{+}\mathit{false} \vee \langle\alpha\rangle\langle\beta\rangle\mathrm{P}$,

(g) $[\alpha\cup\beta]^{+}P \equiv [\alpha]^{+}P \wedge [\beta]^{+}P$,
(h) $\langle\alpha\cup\beta\rangle^{+}P \equiv \langle\alpha\rangle^{+}P \vee \langle\beta\rangle^{+}P$,
(i) $[\alpha]^{+}(P\wedge Q) \equiv [\alpha]^{+}P \wedge [\alpha]Q$,
(j) $\langle\alpha\rangle^{+}(P\vee Q) \equiv \langle\alpha\rangle^{+}P \vee \langle\alpha\rangle Q$,
(k) $[\alpha^{*}](P\supset\langle\alpha\rangle P) \supset (P\supset\langle\alpha^{*}\rangle^{+}\mathit{false})$,

Proof. All follow quite easily from the definitions. We omit the proofs. ∎

Note the (seemingly paradoxical) equivalent assertions:

$$[\alpha]^{+}\mathit{false} \equiv [\alpha]^{+}\mathit{true} \wedge [\alpha]\mathit{false}.$$

Define $\alpha^{0} =_{df}$ *true?* (i.e. the identity relation), and $\alpha^{n+1} =_{df} \alpha;\alpha^{n}$.

Theorem 1. $\mathfrak{J}\models\langle\alpha^{*}\rangle^{+}\mathit{false}$ iff $\forall n\geq 0\ \ \mathfrak{J}\models\langle\alpha^{n}\rangle^{+}\mathit{true}$.

A rigorous proof of this theorem is given in the Appendix (Section 6). Intuitively, the right hand side implies that arbitrarily long computation sequences can be found in $t(\alpha^{*},\mathfrak{J})$, which must therefore be infinite, implying the left hand side. Conversely, if $t(\alpha^{*},\mathfrak{J})$ contains an infinite path then $t(\alpha^{n},\mathfrak{J})$ must either contain an infinite path or a halt node.

Corollary 1. $\mathfrak{J}\models[\alpha^{*}]^{+}\mathit{true}$ iff $\exists n\geq 0$ such that $\mathfrak{J}\models[\alpha^{n}]^{+}\mathit{false}$.

At this point we will start talking about the specific universe of arithmetic N (see [12]), remarking that for the universe A of arithmetic with uninterpreted function and predicate symbols (augmented arithmetic), all results of this paper would also hold. We can now rephrase the results of the previous Theorem and Corollary as

$$\models_{N} [\alpha^{*}]^{+}\mathit{true} \equiv \exists n[\alpha^{n}]^{+}\mathit{false},$$
$$\models_{N} \langle\alpha^{*}\rangle^{+}\mathit{false} \equiv \forall n\langle\alpha^{n}\rangle^{+}\mathit{true}.$$

An equivalent but more comprehensible description of the behavior of the plus-modalities on α^{*} is given by the following Theorem and Corollary, the proofs of which we omit:

Theorem 2. $\models_{N} \langle\alpha^{*}\rangle^{+}\mathit{false} \equiv \forall n\langle\alpha^{n}\rangle\mathit{true} \vee \langle\alpha^{*}\rangle\langle\alpha\rangle^{+}\mathit{false}$.

Corollary 2. $\models_{N} [\alpha^{*}]^{+}\mathit{true} \equiv \exists n[\alpha^{n}]\mathit{false} \wedge [\alpha^{*}][\alpha]^{+}\mathit{true}$.

Theorem 2 states that a divergence in α^{*} is due either to being able to "run" α for as many times as you wish (which by Koenig's Lemma is equivalent to being able to run α for ever), and this we might term diverging for *global* reasons, or to being able to run α for a certain number of times and then have α itself diverge, terming this diverging for *local* reasons. Corollary 2 states that α^* is divergence-free if there is a limit on the "depth" we can go to by doing α, and that furthermore α itself does not diverge in the process.

We recall the fact (mentioned in [12]) that (augmented) first order arithmetic is expressive for (augmented) DL. We extend that result to:

Theorem 3. First order arithmetic is expressive for DL$^+$; i.e. for every DL$^+$ formula P, there exists a formula F of first order arithmetic such that $\models_N F\equiv P$.

Proof. Use induction on the structure of formulae and programs via the result in [12] and Corollary 2. ∎

We make a remark here which concerns the Algorithmic Logic of the group of Polish researchers initiated by Salwicki [24]. They employ an operator $(\cap\alpha)$, for which $(\cap\alpha)P$ is to be equivalent to $\forall n\langle\alpha^n\rangle P$. Although [24] allows only programs which are deterministic, we can admit this operator into DL (allowing the formula $(\cap\alpha)P$ for any program α and formula P, with the above semantics), and call the resulting logic ADL. We then have:

Theorem 4. ADL is expressive for DL$^+$; i.e. for every DL$^+$ formula P, there exists an ADL-wff F such that $\models_N F\equiv P$.

Proof. Again induction is employed. This time the key fact is Theorem 2 which is rephrased as $\mathcal{J}\models\langle\alpha^*\rangle^+$*false* (in DL$^+$) iff $\mathcal{J}\models(\cap\alpha)$*true* $\wedge\ \langle\alpha^*\rangle$"$\langle\alpha\rangle^+$*false*" (in ADL; where "$\langle\alpha\rangle^+$*false*" stands for the ADL formula equivalent to $\langle\alpha\rangle^+$*false*, which exists by the inductive hypothesis). ∎

In [9] an arithmetically complete axiomatization of ADL has been exhibited.

We would like to pause here and pose an important open problem (pointed out to us by M.J. Fischer), concerning the expressive power of DL$^+$:

Is it the case that for every DL$^+$-wff P *there exists a DL-wff* Q *such that* $\models P\equiv Q$?

We strongly conjecture that the answer to this question is *no*, i.e. that DL$^+$ is strictly stronger than DL in expressive power. Meyer [20] has shown that if array assignments of the form $F(x)\leftarrow y$ are allowed, then the answer is *yes*, i.e. DL$^+$ is no more expressive than DL. However, the proof in [20], and the manner in which it uses array assignments in DL to make possible expressing the presence of a divergence, further convinces us that with simple assignments the answer is nevertheless *no*. Another version of this question, is in the case of propositional DL (PDL, see Fischer and Ladner [7]), where we can similarly define PDL$^+$. It can be shown that PDL$^+$ is strictly more expressive than PDL. Other related questions and observations can be found in [20].

We now augment P to P^+, by adding the following axioms and rules:

(J) $[x\leftarrow E]^+\mathit{true}$,
(K) $[Q?]^+\mathit{true}$,
(L) $[\alpha;\beta]^+\mathit{true} \equiv [\alpha]^+[\beta]^+\mathit{true}$,
(M) $[\alpha\cup\beta]^+\mathit{true} \equiv [\alpha]^+\mathit{true} \wedge [\beta]^+\mathit{true}$,

(N) $[\alpha]^+P \equiv [\alpha]^+\mathit{true} \wedge [\alpha]P$,

(O) Finiteness
$$\frac{P(n+1)\supset[\alpha]^+P(n)\ ,\ \neg P(0)}{P(n)\supset[\alpha^*]^+\mathit{true}}$$

where n does not appear in α.

(P) Divergence
$$\frac{P\supset\langle\alpha\rangle^+P}{P\supset\langle\alpha^*\rangle^+\mathit{false}}$$

The soundness of the axioms follows from the above discussion, and the soundness of rules (P) and (O) can be shown to follow from Theorems 1 and Corollary 2 respectively. We proceed to show what we might call the "completeness" of each of the last two rules, followed by box and diamond completeness theorems for DL$^+$, and then present our main result. For these purposes denote P^+ without rules (I) and (P) by $P^+[\,]$, and without rules (H) and (O) by $P^+\langle\rangle$. As in [12], we will concern ourselves with the proofs of the completeness directions of the theorems only.

Lemma 2. If $\models R\supset[\alpha^*]^+\mathit{true}$, then there exists a formula of arithmetic P(n) with free variable n, such that the premises of rule (O) and $R\supset\exists nP(n)$ are all valid.

Proof. By expressiveness of arithmetic, $[\alpha^n]^+\mathit{false}$ is expressible as such a P(n). By Corollary 2, $\models R\supset\exists nP(n)$. Noting that $P(0)\equiv\mathit{false}$, we observe the trivial validity of the premises of (O). ∎

Theorem 4 (DL$^+$ Box Completeness Theorem). For any formulae of arithmetic R and Q and program α

$$\models_N R\supset[\alpha]^+Q \quad \text{iff} \quad \vdash_{P^+[\,]} R\supset[\alpha]^+Q.$$

Proof. Assume $\models_N R\supset[\alpha]^+Q$, which really asserts that both $\models_N R\supset[\alpha]Q$ and $\models_N R\supset[\alpha]^+\mathit{true}$ hold. By the DL Box Completeness Theorem of [12] we have $\vdash_{P^+[]} R\supset[\alpha]Q$. We shall show that $\vdash_{P^+[]} R\supset[\alpha]^+\mathit{true}$, and axiom (N) will serve to combine the results. This follows, however, by induction on the structure of α (using Lemma 2 for justifying the application of rule (O) when α is β^*), from which we obtain $P(n)\supset[\alpha^*]^+\mathit{true}$, and then $R\supset[\alpha^*]^+\mathit{true}$. ■

Lemma 3. If $\models R\supset\langle\alpha^*\rangle^+\mathit{false}$, then there exists a formula P of arithmetic such that the premise of rule (P) and $R\supset P$ are both valid.

Proof. Assume $\models R\supset\langle\alpha^*\rangle^+\mathit{false}$. We will exhibit here a situation similar to that occurring with the rule of Invariance; in [22] it was implicitly shown that both $[\alpha^*]P$ and $\langle(\alpha^-)^*\rangle R$ could be taken as invariants, satisfying those conditions which guaranteed that the rule could be applied. Here too, both a "strongest $\langle\rangle^+$ consequent" and a "weakest $\langle\rangle^+$ antecedent" will be shown to satisfy the requirements of the Lemma. The latter is simply $\langle\alpha^*\rangle^+\mathit{false}$ which trivially satisfies the requirements. The former is a little more subtle. Intuitively what we will construct is the predicate P which is *true* exactly in those states which lie on an infinite path of "computation" which started in a state satisfying R (by assumption there is at least one such path for every such starting state). Take P to be an arithmetical equivalent of $\langle(\alpha^-)^*\rangle R\wedge\langle\alpha^*\rangle^+\mathit{false}$ (recall that $\mathcal{I}(\alpha^-)\mathcal{J}$ iff $\mathcal{J}\alpha\mathcal{I}$). As first order arithmetic can be shown to be expressive for DL$^+$ augmented with the *converse* $(^-)$ operator on programs, this equivalent exists. $\langle(\alpha^-)^*\rangle R$ states that the present state is on a path from R via α^*, and $\langle\alpha^*\rangle^+\mathit{false}$ makes sure that we are on a path with a possible infinite computation. First we observe that $R\supset\langle(\alpha^-)^*\rangle R$, and by the assumption also $R\supset\langle\alpha^*\rangle^+\mathit{false}$. Thus $R\supset P$ is valid. We are left with having to show $P\supset\langle\alpha\rangle^+P$. This can be seen to follow directly from the easily checked validities

1. $\langle(\alpha^-)^*\rangle R\supset[\alpha]\langle(\alpha^-)^*\rangle R$,
2. $\langle\alpha^*\rangle^+\mathit{false}\equiv\langle\alpha\rangle^+\langle\alpha^*\rangle^+\mathit{false}$, and
3. $([\alpha]U\wedge\langle\alpha\rangle^+V)\supset\langle\alpha\rangle^+(U\wedge V)$

   (taking U to be $\langle(\alpha^-)^*\rangle R$ and V to be $\langle\alpha^*\rangle^+\mathit{false}$)

(To check 1, use the validities $W\supset[\alpha]\langle\alpha^-\rangle W$ and $\langle\beta\rangle\langle\beta^*\rangle R\supset\langle\beta^*\rangle R$, taking W to be $\langle(\alpha^-)^*\rangle R$ and β to be α^-.) ■

Theorem 5 (DL$^+$ Diamond Completeness Theorem). For any formulae of arithmetic R and Q, and program α,

$$\models_N R\supset\langle\alpha\rangle^+Q \quad\text{iff}\quad \vdash_{P^+\langle\rangle} R\supset\langle\alpha\rangle^+Q.$$

Proof. Analogous to the previous theorem, using Lemma 3. ■

Lemma 4. For any DL$^+$ wffs P and Q, and for any program α, if $\vdash_{P^+} P\supset Q$, then

(a) $\vdash_{P^+}[\alpha]P\supset[\alpha]Q$,

(b) $\vdash_{P^+}\langle\alpha\rangle P\supset\langle\alpha\rangle Q$,

(c) $\vdash_{P^+}[\alpha]^+P\supset[\alpha]^+Q$,

(d) $\vdash_{P^+}\langle\alpha\rangle^+P\supset\langle\alpha\rangle^+Q$.

Proof. (a) is obtained by rule (G), as is (b) (with $\neg Q \supset \neg P$); (c) and (d) are obtained similarly with help of axiom (N). ■

Theorem 6 (DL$^+$ Completeness Theorem). For any DL$^+$-wff P,

$$\models_{N} P \text{ iff } \vdash_{\mathcal{P}^+} P.$$

Proof. The framework of the proof follows in the footsteps of our proof of the DL-Completeness Theorem of [12], which employs induction on the number n of modalities in P, and works with a conjunctive normal form. Rather than reproduce it here, we refer the reader to [12], pointing out that Theorem 3 (above) gives us the expressiveness we need, Theorems 4 and 5 provide for the justification of the additional parts of the case n=1, and Lemma 4 is needed for the last stage in which $\vdash_{\mathcal{P}^+} m(\alpha)L(P_2) \supset m(\alpha)P_2$ is established. ■

A subsequent paper [9] provides insight as to the pattern by which we obtain arithmetically complete rules for various modalities applied to α^*, and in the process clarifies the analogy between the aforementioned rules for $[\alpha^*]^+$ and $\langle\alpha^*\rangle^+$, and the $[\alpha^*]$ and $\langle\alpha^*\rangle$ rules of $\mathcal{P}$. Thus, e.g. the invariant assertion method of Floyd/Hoare, which is captured concisely by the rule of Invariance (H), is seen to fall out easily as a special case of a much broader observation.

In [11], the work of this section and of [12] is considerably extended, by adding a recursion operator $\mu X \tau(X)$ to ; , $\cup$ and $*$, together with inference rules for $[\mu X \tau(X)]$, $\langle\mu X \tau(X)\rangle$, $[\mu X \tau(X)]^+$ and $\langle\mu X \tau(X)\rangle^+$. The resulting axiom system is shown to be sound and arithmetically complete. The interesting part occurs when the analogue of e.g. Theorem 2 is attempted for $\mu X \tau(X)$.

4. *Total Correctness and Weakest Preconditions.*

Let us try to clarify the notion of "total correctness" of a nondeterministic program α. In the sequel we will argue that this is a concept which is necessarily dependent on the particular notion of execution of α one has in mind. We will consider $t(\alpha,\mathcal{J})$ and $c(\alpha,\mathcal{J})$ of Section 2.

Since $c(\alpha,\mathcal{J})$ might be an infinite set, a feasible "execution" of α in $\mathcal{J}$ cannot in general be carried out on a machine by calculating $c(\alpha,\mathcal{J})$, choosing one or some of its elements, and proceeding to execute them. Rather, what we need is to choose some method of traversing $t(\alpha,\mathcal{J})$ which will either eventually lead to a halt node (either an f-node or "good" node), or will go on for ever. We envisage four possible such methods:

(1) *Depth first.* At each node choose arbitrarily between possibilities and proceed; stop when a halt node is reached.

(2) *Depth first with backtracking.* Same as (1); backtrack if an f-node is encountered.

(3) *Breadth first.* At each node pursue all possibilities simultaneously; stop when a halt node is reached; if more than one is reached together, choose one arbitrarily.

(4) *Breadth first with ignoring.* Same as (3); ignore all f-leaves (in stopping and in choosing).

We now observe that assuming the existence of at least one final state ($\mathcal{I}\models\langle\alpha\rangle true$), if the machine executing a program α in state $\mathcal{I}$ using these various methods is to reach a final state upon completion, without having diverged or failed, then $t(\alpha,\mathcal{I})$ should adhere to the following table, where "no" means that $c(\alpha,\mathcal{I})$ should be free of failure/divergence elements (f-nodes/infinite paths):

	failure	*divergence*
(1)	no	no
(2)	yes	no
(3)	no	yes
(4)	yes	yes

Thus for example, if we choose method (2), we do not mind having f-leaves in the tree, but do not want any infinite paths to be present.

The importance of the above table is that it illustrates the fact that the notion of total correctness of nondeterministic programs is indeed strongly dependent upon the particular method used. Saying that α is totally correct with respect to R and Q amounts to saying that for any $\mathcal{I}$ such that $\mathcal{I}\models$R, application of the chosen method of execution is guaranteed to lead to a final state $\mathcal{J}$ such that $\mathcal{J}\models$Q. Thus, whenever $\mathcal{I}\models$R, we always want both a final state to exist, and every final state to satisfy Q. Furthermore, whether we require the absence of divergences and/or failures depends upon the execution method used. We arrive, therefore, at the following description of the notion of total correctness of nondeterministic programs, the variants of this concept for the four methods being respectively:

(1) $R \supset (\langle\alpha\rangle true \wedge [\alpha]Q \wedge fl(\alpha) \wedge dv(\alpha))$,
(2) $R \supset (\langle\alpha\rangle true \wedge [\alpha]Q \wedge dv(\alpha))$,
(3) $R \supset (\langle\alpha\rangle true \wedge [\alpha]Q \wedge fl(\alpha))$,
(4) $R \supset (\langle\alpha\rangle true \wedge [\alpha]Q)$,

where $\mathcal{J}\models fl(\alpha)$ iff $c(\alpha,\mathcal{J})$ has no *failures*, and $\mathcal{J}\models dv(\alpha)$ iff $c(\alpha,\mathcal{J})$ has no *divergences*. Adopting DL$^+$, $dv(\alpha)$ is simply $[\alpha]^+ true$, thus total correctness for method (2) can be expressed in DL$^+$ as

$$R \supset ([\alpha]^+Q \wedge \langle\alpha\rangle true).$$

We would like to point out at this stage, that we are quite satisfied with the adequacy of DL$^+$ for capturing the notion of total correctness for methods (2) and (4) (for the latter it is expressible even in DL). One cannot imagine an implementation using breadth first search without ignoring the failure nodes, as in (3). Also, backtracking has become an integral part of depth first search, to the point of people having difficulty in envisioning it without. As we shall see, however, Dijkstra's notion of weakest precondition is really addressed to method (1), and it is, therefore, at this concept that we now direct our attention, showing, quite surprisingly, that DL$^+$ is powerful enough to capture the concept of failing too, and hence the notion of total correctness for all four methods.

In [5] and [6] Dijkstra introduced the concept of weakest precondition:

(*) "The condition that characterizes the set of *all* initial states such that activation will certainly result in a properly terminating happening, leaving the system in a final state satisfying a given post-condition" ([6], p. 16).

This condition was defined to be the value of a function $wp(\alpha,P)$, which we will write as $\{\alpha\}P$, satisfying the five conditions:

(a) $\{\alpha\}false \equiv false$
(b) if $\models P\supset Q$ then $\models \{\alpha\}P \supset \{\alpha\}Q$
(c) $\{\alpha\}P \wedge \{\alpha\}Q \equiv \{\alpha\}(P\wedge Q)$
(d) $\{\alpha\}P \vee \{\alpha\}Q \supset \{\alpha\}(P\vee Q)$
(e) if $\forall i\geqslant 0(P_i\supset P_{i+1})$ then $\{\alpha\}\exists iP_i \equiv \exists j\{\alpha\}P_j$ (continuity).

Dijkstra then introduces a syntax for a programming language, and defines its semantics by specifying $\{\alpha\}P$ for each allowed program α. Specifically, the programs and his definitions are:

empty program $\{false?\}P =_{df} false$,

identity prog. $\{true?\}P =_{df} P$,

assignment $\{x\leftarrow E\}P =_{df} P^E_x$, composition $$\{\alpha;\beta\}P =_{df} \{\alpha\}\{\beta\}P,$$

IF $$\{\bigcup_{i=1}^{n} (Q_i?;\alpha_i)\}P =_{df} (\bigvee_{i=1}^{n} Q_i) \wedge \bigwedge_{i=1}^{n} (Q_i \supset \{\alpha_i\}P)$$

DO $$\{(\bigcup_{i=1}^{n} (Q_i?;\alpha_i))^*;(\bigwedge_{k=1} \neg Q_k)?\}P =_{df} \exists j H_j(P),$$

where $H_0(P) =_{df} P \wedge (\bigwedge_{i=1}^{n} \neg Q_k)$, and

$$H_{j+1}(P) =_{df} H_0(P) \vee \{\bigcup_{i=1}^{n} (Q_i?;\alpha_i)\}H_j(P).$$

At this point we should assume that (a)-(e) are to characterize the notion of *wp*, and that they ought to give rise to a unique "predicate transformer" $\{\alpha\}P$. This, as we shall see (and as observed independently in e.g. [23] and [15]), is not the case. Moreover, there seems to be no further definition to fall back on, if what we want is to understand what *wp*(α,P) is really saying (i.e. which states $\mathcal{I}$ satisfy *wp*(α,P)). The English description (*) is vague, and uses the word "activation", so leaving unspecified which method of activation is being used. The definition of Dijkstra's programming language *uses wp*, and certainly does not *define* it, and thus cannot be of much help.

We first set out to find something which satisfies (a)-(e) for regular α's. Inspection shows that (c) fails for $\langle\alpha\rangle P$ and (e) for $[\alpha]P$. (Example of latter: take α to be $x\leftarrow 0;(x\leftarrow x+1)^*$ and P_i to be $x\leqslant i$.) This, however should not cause alarm, because (a)-(e) are required to hold for the final product satisfying (*), which is to have the always ("will *certainly*") and the sometimes ("*will* certainly") properties. An attempt to take $\{\alpha\}P$ to be $[\alpha]P \wedge \langle\alpha\rangle P$ (equivalently $[\alpha]P \wedge \langle\alpha\rangle$*true*) will result in (c) holding, but (e) still failing (same example). This observation explains our remark in [12] in which we incorrectly claimed *wp*(α,P) to be $[\alpha]P \wedge \langle\alpha\rangle$*true*, which was made solely on the basis that this construct satisfies the axioms which appeared in [5] ((a)-(d)), without referring to the later addition (e) of [6]. The reason (e) fails for $[\alpha]$ (and for that matter fails for $[\alpha]P \wedge \langle\alpha\rangle$*true*) is rooted in what [6] calls "unbounded nondeterminism", which in this case is a misleading term: α^* can, as remarked, be viewed as a tree with *finite* outdegree, but can have infinitely many final states, by virtue of being able to apply α infinitely often. Outlawing this situation is brought about by further asserting the absence of divergences! Thus we propose to add to our suggestion $[\alpha]^+$*true*, arriving at

$$\{\alpha\}P =_{df} [\alpha]^+P \wedge \langle\alpha\rangle true.$$

Lemma. Conditions (a)-(e) are satisfied by $\{\alpha\}P$.
Proof. Omitted.

Thus it would seem that our task is completed, as would indeed be the case if *wp* was defined to be any predicate transformer satisfying (a)-(e). We might note here that the result of Wand [25] strengthens our argument. Wand has shown that if a program is *defined* to have the property of allowing only a finite number of possible out-states for every initial state (which in our case amounts to asserting $[\alpha]^{+}true$), then a functional $F:2^{U} \rightarrow 2^{U}$ is equivalent to $[\alpha]P \wedge \langle\alpha\rangle true$ for some such α *iff* (a)-(e) hold for F (that is substitute F for $\{\alpha\}$ in (a)-(e)). This result is essentially showing that our $\{\alpha\} = [\alpha]^{+}P \wedge \langle\alpha\rangle true$ is indeed the "greatest" predicate transformer satisfying these axioms. However, the fact is that there is more, in Dijkstra's description, than just the axioms. Let us take a look at the definition of his language. We notice that $\{\alpha;\beta\}P=\{\alpha\}\{\beta\}P$ does not hold for our definition of $\{\alpha\}P$, and neither do the "equalities" for IF and DO. (Parenthetically, we might note that, contrary to what is implied in [15], the importance of having e.g. the property $wp(\alpha;\beta,P) = wp(\alpha,wp(\beta,P))$ hold, should not at all be overestimated. This property is important for easing the construction of the weakest precondition of a given P with respect to a given α, only if the wp construct is the only thing you have! In our case the fact that $\{\alpha\}P$ does not enjoy the property is irrelevant, because $\{\alpha\}P$ is defined as the conjunction of two constructs which *do* (namely $\langle\alpha\rangle true$ and $[\alpha]^{+}P$). Constructing $\{\alpha\}P$ in a concrete situation would proceed, therefore, by constructing the two components in the natural way using this property for both!) These observations call for extra analysis.

Suppose we were to attempt to capture the *third* class of members of $c(\alpha,\mathcal{J})$, namely the failures, by adding a *failure-state* (say $\perp'$) to U^{+}. Despite the fact that Dijkstra's axioms do not require this, his informal discussion of failure (referred to as abortion) indicates that failure is considered not to be proper termination, whence if we are to capture what Dijkstra had in mind by wp (as opposed to what he axiomatized) we will need to include $\perp'$ in our reckoning of what constitutes a binary relation abstraction of a program. We can do this as we did for $\perp$: $(\mathcal{J},\perp')$ is in the binary relation for α when $t(\alpha,\mathcal{J})$ contains a failure node. We now give a way of expressing, in DL^{+}, the formula $fl(\alpha)$ whose truth in state $\mathcal{J}$ asserts that $t(\alpha,\mathcal{J})$ contains no failure node.

$$\begin{aligned}
fl(x\leftarrow E) &\equiv true,\\
fl(P?) &\equiv true,\\
fl(\alpha\cup\beta) &\equiv fl(\alpha) \wedge fl(\beta),\\
fl(\alpha;\beta) &\equiv fl(\alpha) \wedge tr(\alpha,\beta) \wedge [\alpha]fl(\beta),\\
fl(\alpha^{*}) &\equiv [\alpha^{*}]fl(\alpha).
\end{aligned}$$

In turn we need to define the "transition" predicate $tr(\alpha,\beta)$, which expresses the hope that if α leads to a state from which β has no chance of eventual success or divergence then α may be resumed. (Such resumption is possible with programs such as γ^*, where one might execute γ some number of times, try to execute β, fail immediately, and so execute γ again. This is precisely the behavior one would expect in the interpretation of the DL translation of *while* P *do* α as $(P?;\alpha)^*;\neg P?$.)

$$\begin{aligned}
tr(\alpha,\beta) &\equiv [\alpha]\langle\beta\rangle^{+}true \quad \text{for } \alpha \text{ an assignment or test}\\
tr(\gamma;\delta,\beta) &\equiv [\gamma]tr(\delta,\beta)\\
tr(\gamma\cup\delta,\beta) &\equiv tr(\gamma,\beta) \wedge tr(\delta,\beta)\\
tr(\gamma^*,\beta) &\equiv [\gamma^*]\langle\beta\cup\gamma\rangle^{+}true
\end{aligned}$$

Let us define now $\{\{\alpha\}\}P =_{df} [\alpha]^{+}P \wedge \langle\alpha\rangle true \wedge fl(\alpha)$.

Lemma. Conditions (a)-(e) are satisfied by $\{\{\alpha\}\}P$.
Proof. Omitted.

It can be shown (quite tediously though) that the equalities in Dijkstra's definition of the guarded command language above *also* hold for $\{\{\alpha\}\}P$. Thus $\{\{\alpha\}\}P$ *exactly* expresses $wp(\alpha,P)$, namely the weakest condition guaranteeing correct termination when execution method (1) is adopted. Now, given that fl can be expressed using DL^+, and given that we have completely axiomatized DL^+, we infer that we have also completely axiomatized $\{\{\alpha\}\}P$. So we have given an arithmetically complete axiomatization of Dijkstra's notion of weakest precondition.

These remarks, combined with the elegance of the logic developed in Section 3, seem, even without any additional backing up, to point to the unavoidable conclusion that reasoning about programs should be carried out using well-defined primitive basic concepts (such as $[\alpha]P$, $\langle\alpha\rangle P$, $[\alpha]^{+}true$, $fl(\alpha)$ etc.) as the building blocks, from which other more complex notions can be constructed. We are strongly against the approach implicit in Dijkstra's work, in which the basic construct (wp) and the properties required of it, appear to obscure the simple parts of which it consists. We are further against the attempt to define the *semantics* of a language using a complex (execution-method dependent in this case) notion. Recently deBakker [2], deRoever and Plotkin (unpublished) have gone to great efforts in trying to find the appropriate denotational semantics which would correspond to what essentially is Dijkstra's definition, using predicate transformers, of the semantics of a recursive programming language (with the explicit addition of an *if then else* construct). In our opinion, not only have they not fully solved the problem of tying up a denotational semantics to a total-correctness-based semantics (the same process remains to be carried out for the other three execution methods, for their solution implicitly assumes the need for outlawing both divergences and failures), but also the "top down" viewpoint of having to "apologize" for defining a language with the aid of wp, by conjuring up a suitable ordering on the domain which gives rise to an equivalent denotational definition, is inferior to the much healthier "bottom up" approach. The latter consists of first defining the semantical objects (states and binary relations), and only then introducing the logical language and assigning meaning in the manner of Tarski. At this point the truth of the formulae of the logical language has already been determined (and in a plausible way!), and the "axioms" of Dijkstra's definition can then be verified as mere theorems. This formalizes what happens when a semanticist is confronted with those axioms; he attempts to verify their agreement with his intuition about the behavior of programs. In contrast, deBakker, deRoever and Plotkin (in this case) start from the axioms and assign semantics that are faithful to those axioms.

5. *Other Work.*

Some of the points made in the previous section appear also (either explicitly or implicitly) in Hoare [15]. He writes $\alpha\ a\ P$ and $\alpha\ e\ P$ (*allows* and *ensures*) for $\langle\alpha\rangle P$ and $[\alpha]P$ respectively and notes the duality of $\langle\rangle$ and $[]$. (The notation was interestingly confusing for us, since $\langle\rangle$ and $[]$ correspond to $\exists$ and $\forall$, or *e* and *a* respectively, not to *a* and *e*!) More significant is his definition of $b(\alpha)$, which in state $\mathcal{I}$ asserts the finiteness of the members of $c(\alpha,\mathcal{I})$, and which, if we take the liberty of rewriting for our α^*, is $\vee_{n\geqslant 0}(\alpha^n\ e\ \mathit{false})$, or as we would write, $\exists n[\alpha^n]\mathit{false}$, which should now be viewed in the light of Section 3. Hoare observes that $\{\alpha\}P = [\alpha]^+P \wedge \langle\alpha\rangle \mathit{true}$ does not satisfy "the basic axiom for program composition", and proceeds to describe explicitly in his logic of traces, that condition $f(\alpha)$ (similar to our $fl(\alpha)$) which guarantees the non-existence of failures ("blind alleys"), and shows that $wp(\alpha,P)$ defined as $\{\alpha\}P \wedge f(\alpha)$, satisfies that too.

Manna [17] uses essentially quantification over states to spell out a variety of properties of deterministic and nondeterministic programs. His programs are many-valued functions of states, i.e. $\alpha(\mathcal{I}) \subset U$. $\mathcal{I} \models \langle\alpha\rangle P$ is termed "α *totally* $\exists$*-correct* wrt P in $\mathcal{I}$", and is written $\exists\mathcal{J}(\mathcal{J}\in\alpha(\mathcal{I}) \wedge P(\mathcal{I},\mathcal{J}))$ (note the similarity to $\exists\mathcal{J}(\mathcal{I}\alpha\mathcal{J} \wedge \mathcal{J}\models P)$), and $\mathcal{I} \models [\alpha]P$ termed "α *partially* $\forall$*-correct* wrt P in $\mathcal{I}$" and written $\forall\mathcal{J}(\mathcal{J}\in\alpha(\mathcal{I}) \supset P(\mathcal{I},\mathcal{J}))$. "*Total equivalence*" of α and β in state $\mathcal{I}$ is written "$\alpha(\mathcal{I})$ and $\beta(\mathcal{I})$ are not empty, and $\alpha(\mathcal{I})=\beta(\mathcal{I})$", the last conjunct being inexpressible in DL without the use of state quantifiers. [17] then proceeds to express the above and other properties using (here we translate for clarity into our own terminology) the primitive $[\alpha]P$ and second order quantifiers; in particular it is noted (perhaps the earliest mention of this fact) that $\langle\alpha\rangle P \equiv \neg[\alpha]\neg P$, and that "*totally* $\forall$*-correct*" is $\neg[\alpha]\mathit{false} \wedge [\alpha]P$ or $\langle\alpha\rangle \mathit{true} \wedge [\alpha]P$. Manna [17] then augments U with an undefined state ∞, and requires that the programs map any state into a *non-empty* subset of $U\cup\{\infty\}$, thus as is the case with [1] and [23] (see below), $(\mathcal{I},\infty)$ is added to the program not only in the presence of a divergence, but also in the case of failure, and hence differs from DL^+. However, [17] has four concepts of correctness analogous (and not identical, due to the above difference) to our modalities:

partially ∃*-correct* (written $\exists\mathcal{J}(\mathcal{J}\in\alpha(\mathcal{I}) \wedge (\mathcal{J}\neq\infty\supset P(\mathcal{I},\mathcal{J}))))$
totally ∃*-correct* $(\exists\mathcal{J}(\mathcal{J}\in\alpha(\mathcal{I}) \wedge \mathcal{J}\neq\infty \wedge P(\mathcal{I},\mathcal{J})))$
partially ∀*-correct* $(\forall\mathcal{J}((\mathcal{J}\in\alpha(\mathcal{I}) \wedge\mathcal{J}\neq\infty) \supset P(\mathcal{I},\mathcal{J})))$
totally ∀*-correct* $(\forall\mathcal{J}(\mathcal{J}\in\alpha(\mathcal{I}) \supset (\mathcal{J}\neq\infty \wedge P(\mathcal{I},\mathcal{J}))))$

analogous to $\langle\alpha\rangle^{+}P$, $\langle\alpha\rangle P$, $[\alpha]P$, and $[\alpha]^{+}P$ respectively.

The double duality of these concepts is proved, and other properties are expressed in a second order language using these four as basics. The aformentioned difference in the use of ∞, comes to the surface in the remark of [17] that $\langle\alpha\rangle^{+}$*true* is valid for every α, a fact which is not true in DL^{+}. However, this clean description of the four basic concepts of nondeterministic programs, formulated already in 1970 (and brought to our attention by N. Dershowitz), strengthens our confidence in the manner in which we have constructed DL^{+}.

In deRoever [23], who uses explicit quantification over states, we find in essence the remark that $[\alpha]P\wedge\langle\alpha\rangle$*true* satisfies (a)-(d). Also [23] proceeds to describe the *wp* which satisfies (e) as well as Dijkstra's definitions of (what boils down to) ; , ∪ and *. This is done, as we have remarked, by using ⊥ to express *both* divergences *and* failures (respectively in the words of [23] with notation adjusted: "if $\{\mathcal{J}|(\mathcal{I},\mathcal{J})\in\alpha\}$ is infinite then $(\mathcal{I},\perp)\in\alpha$" and "I feel free to switch from a relation α as a subset of values of some cartesian product U×U, to a function α from $U\cup\{\perp\}$ to nonempty subsets of $U\cup\{\perp\}$, made total by using $\{\perp\}$ as the value for $\alpha(\mathcal{I})$ in case .... $\{\mathcal{J}|(\mathcal{I},\mathcal{J})\in\alpha\}$ is empty"). As remarked, this approach eliminates the possibility of using ∪ to branch conditionally, so an *if then else* is added explicitly to the language! Of course, in this setting $wp(\alpha\cup\beta,P) = wp(\alpha,P) \wedge wp(\beta,P)$ because *wp* enables one to be able to choose either, and not have to backtrack, but on the other hand $wp($*if* R *then* α *else* $\beta,P) = wp(R?;\alpha,P) \vee wp(\neg R?;\beta,P)$. [23] also proves that the Egli-Milner ordering on programs over U^{+} results in $wp(\alpha,P)$ being continuous in α, a result which it is possible to show holds for $\{\alpha\}P$ too.

Turning to deBakker [1], the part of this work relevant to weakest preconditions, differs in an important respect from that of [23]. deBakker uses ∪ and *if then else* as two independent constructs, and does not have our notion of tests. This eliminates all possibility of failure, making deBakker's task simpler than Dijkstra's or ours. In another important respect his work parallels ours; he states explicitly "... with the convention that $\mathcal{I}\alpha\perp$ holds iff some computation sequence specified by α does not terminate properly .... either the computation sequence is infinite, or that one of the elementary actions .... is undefined at some intermediate state". Interestingly, [1] reaches the conclusion (as did we independently) that it is very helpful to define ⊥ such that $\perp\not\models P$ for all P. What he really does is to restrict his predicates to those that are *false* in ⊥. Another noteworthy remark about [1] is his $e(\alpha)$, which is defined to be *true* in $\mathcal{I}$ iff $(\mathcal{I},\perp)$ is not in α, and thus in this light, the observation in [1] that e.g. $e(\alpha;\beta) = e(\alpha) \wedge [\alpha]e(\beta)$ amounts to our easily checked equivalence $[\alpha;\beta]^{+}$*true* $\equiv [\alpha]^{+}$*true* $\wedge [\alpha][\beta]^{+}$*true* (see Lemma 1(e)).

Another paper describing a logic which allows nondeterministic programs, and which is supplied with a complete axiom system, is Harel, Pnueli and Stavi [13], in which the nondeterministic assignment $x \leftarrow \in x' P(x,x')$ can be used to simulate general $\cup$ and $*$. However, although $[\alpha]P$ and $\langle\alpha\rangle P$ can essentially be expressed (and by the completeness theorem proved), as well as certain combinations of them, no mechanism is provided for reasoning about divergencies, and hence only total correctness for method (4) can be captured.

6. Appendix

We give here a rigorous proof of Theorem 1. An interesting aspect of the proof is the way in which it makes use of a DL version of Koenig's lemma. While the reader may have been convinced by our informal justification of Theorem 1, the apparent difficulty of proving it carefully suggests that the "intuitively obvious" in this case needs to be approached with caution unless one is willing to accept it as axiomatic.

The theorem asserts that α^* can reach $\perp$ if and only if for any number of iterations of α either a terminating state is reached or a divergence is encountered along the way. The "only if" part should be obvious, but the "if" part appears to depend on Koenig's lemma and the fact that if α doesn't diverge then α can reach only finitely many states, which is true of our particular nondeterministic programming language but not true of such assignments as n:=?.

Since we can state an extended version of Koenig's lemma in dynamic logic, we can increase the rigor of the proof of Theorem 1 by using this extended version. First we need a notion of independent programs. We say that α and β (treated as binary relations) are *independent* when

(i) $\alpha;\beta = \beta;\alpha$ (order independence), and
(ii) $\mathcal{I}\alpha\mathcal{I}'\beta\mathcal{I}'' \wedge \mathcal{I}\alpha\mathcal{J}'\beta\mathcal{I}'' \supset \mathcal{I}'=\mathcal{J}'$, and similarly with α,β interchanged (information independence).

The first condition implies that for every computation $\mathcal{I}\alpha\mathcal{I}'\beta\mathcal{I}''$ there is a computation $\mathcal{I}\beta\mathcal{J}'\alpha\mathcal{I}''$, and similarly with α,β interchanged. The second condition asserts that if we know what state $\alpha;\beta$ went to then we know what state α went to in the process, i.e. the trajectory information can be recovered from the transition information, so that β cannot destroy information supplied by α.

Second, we need a generalization of the notion of "there exist infinitely many." We use $[\alpha]^i P$ to mean that P holds in all but less than i of the states reachable by α.

Hence $[\alpha]^{\infty}P$ means that only finitely many of the states accessible via α can fail to satisfy P. So if $[\alpha]^{\infty}\mathit{false}$ holds in state $\mathcal{I}$ then there are only finitely many states $\mathcal{J}$ satisfying $\mathcal{I}\alpha\mathcal{J}$, i.e. α has only finitely many transitions possible from this state.

Extended Koenig's Lemma (EKL). Let α, β be independent programs. Then

$$[\beta]^{\infty}\mathit{false} \supset (\langle\alpha\rangle^{\infty}\langle\beta\rangle P \supset \langle\beta\rangle\langle\alpha\rangle^{\infty}P).$$

Proof. It suffices to show that the formula holds in an arbitrarily chosen state $\mathcal{I}$. Suppose then that $\mathcal{I}\models[\beta]^{\infty}\mathit{false}$ and $\mathcal{I}\models\langle\alpha\rangle^{\infty}\langle\beta\rangle P$. Let $V = \{\mathcal{I}'|\mathcal{I}\alpha\mathcal{I}' \wedge \mathcal{I}'\models\langle\beta\rangle P\}$. Then since $\mathcal{I}\models\langle\alpha\rangle^{\infty}\langle\beta\rangle P$, V must be infinite. Now let $f:V\to U$ (U is the universe of states) be a function assigning to each element of V an element $\mathcal{I}''$ such that $\mathcal{I}'\beta\mathcal{I}''$ and $\mathcal{I}''\models P$. The construction of V ensures that such an $\mathcal{I}''$ exists for every $\mathcal{I}'$. Since $\mathcal{I}\alpha\mathcal{I}'$ for each $\mathcal{I}'\in V$, property (ii) of independent programs ensures that f is 1-1. Finally let W = $\{(\mathcal{J}',f(\mathcal{I}'))|\mathcal{I}'\in V \wedge \mathcal{I}\beta\mathcal{J}'\alpha f(\mathcal{I}')\}$. Property (i) of independent programs ensures that a $\mathcal{J}'$ can be found for every $f(\mathcal{I}')$, and so the 1-1-ness of f ensures $|W|\geqslant|V|$, so that W must be infinite too. But only finitely many distinct states may appear in the first coordinate of W since $\mathcal{I}\models[\beta]^{\infty}\mathit{false}$. Hence some value $\mathcal{J}'$ of the first coordinate must appear infinitely often in W. Hence $\mathcal{J}'\models\langle\alpha\rangle^{\infty}P$. But $\mathcal{I}\beta\mathcal{J}'$, so $\mathcal{I}\models\langle\beta\rangle\langle\alpha\rangle^{\infty}P$. ■

(It is possible to strengthen the second $\supset$ of EKL to $\equiv$, but we do not use this fact here. There is also a more general statement of the lemma that caters for other cardinals besides ∞, but we do not need this either.)

In addition to the validities of Lemma 1 we need the validity:

$$[\alpha]^{+}\mathit{true} \supset [\alpha]^{\infty}\mathit{false}.$$

Theorem 1. $\mathcal{I}\models\langle\alpha^{*}\rangle^{+}\mathit{false}$ iff $\forall n\geqslant 0$ $\mathcal{I}\models\langle\alpha^{n}\rangle^{+}\mathit{true}$.

Proof. (=>) We prove $\langle\alpha^{*}\rangle^{+}\mathit{false} \supset \langle\alpha^{n}\rangle^{+}\mathit{true}$ by induction on n. When n=0 the result follows trivially. If we now take as our induction hypothesis the result to be proved we get

$$\begin{aligned}
&\langle\alpha^{*}\rangle^{+}\mathit{false}\\
&\supset \langle\alpha^{0}\rangle\mathit{false} \vee \langle\alpha\rangle^{+}\langle\alpha^{*}\rangle^{+}\mathit{false}\\
&\supset \langle\alpha\rangle^{+}\langle\alpha^{n}\rangle^{+}\mathit{true}\\
&\supset \langle\alpha^{n+1}\rangle^{+}\mathit{true}.
\end{aligned}$$

(<=) For this part of the proof we express "for all $n\geqslant 0$ $\mathcal{I}\models\langle\alpha^{n}\rangle^{+}\mathit{true}$" as "$\mathcal{I}\models[n{:=}?]\langle\alpha^{n}\rangle^{+}\mathit{true}$" so that we can conduct the whole argument within dynamic logic. We take "n:=?" to be a program setting n to a nondeterministically chosen natural number, so that $[n:=?]$ expresses $\forall n$. Clearly we have $[n:=?]P \supset \langle n:=? \rangle^{\infty}P$. The argument assumes that α does not itself use n in any way.

$$
\begin{aligned}
& [n:=?]\langle \alpha^n \rangle^+ \mathit{true} \\
\supset\ & [n:=?]\langle \alpha^{n+1} \rangle^+ \mathit{true} \\
\supset\ & [n:=?]\langle \alpha \rangle^+ \langle \alpha^n \rangle^+ \mathit{true} \\
\supset\ & [n:=?](\langle \alpha \rangle^+ \mathit{false} \vee \langle \alpha \rangle \langle \alpha^n \rangle^+ \mathit{true}) \\
\supset\ & \langle \alpha \rangle^+ \mathit{false} \vee [n:=?]\langle \alpha \rangle \langle \alpha^n \rangle^+ \mathit{true})
\end{aligned}
$$

Now if $\langle \alpha \rangle^+ \mathit{false}$ holds then so does $\langle \alpha^* \rangle^+ \mathit{false}$, and we are done. Otherwise if $\langle \alpha \rangle^+ \mathit{false}$ does not hold (i.e. $[\alpha]^+ \mathit{true}$ holds) then we must have

$$
\begin{aligned}
& [\alpha]^+ \mathit{true} \wedge [n:=?]\langle \alpha \rangle \langle \alpha^n \rangle^+ \mathit{true} \\
\supset\ & [\alpha]^{\infty} \mathit{false} \wedge [n:=?]\langle \alpha \rangle \langle \alpha^n \rangle^+ \mathit{true} && \text{(Lemma 1(l))} \\
\supset\ & [\alpha]^{\infty} \mathit{false} \wedge \langle n:=? \rangle^{\infty} \langle \alpha \rangle \langle \alpha^n \rangle^+ \mathit{true} && \text{($\exists$ infinitely many natural numbers)} \\
\supset\ & \langle \alpha \rangle \langle n:=? \rangle^{\infty} \langle \alpha^n \rangle^+ \mathit{true} && \text{(EKL)} \\
\supset\ & \langle \alpha \rangle [n:=?]\langle \alpha^n \rangle^+ \mathit{true}. && (m<n \supset (\langle \alpha^n \rangle^+ \mathit{true} \supset \langle \alpha^m \rangle^+ \mathit{true}))
\end{aligned}
$$

Thus we have $[n:=?]\langle \alpha^n \rangle^+ \mathit{true} \supset \langle \alpha \rangle [n:=?]\langle \alpha^n \rangle^+ \mathit{true}$. But this is of the form $P \supset \langle \alpha \rangle P$, so applying Lemma 1(k) we get $[n:=?]\langle \alpha^n \rangle^+ \mathit{true} \supset \langle \alpha^* \rangle^+ \mathit{false}$. ■

7. *Acknowledgments.*

The early stages of our thinking about nondeterminism, which finally led to the formulation of DL$^+$, were strongly influenced by numerous discussions with Nachum Dershowitz, who has also helpfully commented on various parts of the paper. Adi Shamir supplied the idea behind the rule of Finiteness. Edsger W. Dijkstra pointed out an error in a previous version of Section 4. We have received valuable feedback from Albert R. Meyer.

8. *References*

[1] deBakker, J.W. Semantics and Termination of Nondeterministic Recursive Programs. In *Automata, Languages and Programming*, Edinburgh. 435-477, 1976..

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

[3] Basu, S. K. and R. T. Yeh. Strong Verification of Programs. IEEE Trans. Software Engineering, SE-1, 3, 339-345. Sept. 1975.

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

[5] Dijkstra, E. W. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. CACM vol 18, no.8. 1975

[6] Dijkstra, E. W. *A Discipline of Programming.* Prentice-Hall. 1976

[7] Fischer, M.J. and R.L Ladner. Propositional Modal Logic of programs. Proc. 9th Ann. ACM Symp. on Theory of Computing, Boulder, Col., May 1977.

[8] Green, C. Cordell. The Application of Theorem Proving to Question-Answering Systems. Stanford University Computer Science Department Report CS-138. 1969.

[9] Harel, D. Arithmetical Completeness in Logics of Programs. Submitted for publication.

[10] Harel, D. On the Correctness of Regular Deterministic Programs; A Unified Survey. Submitted for publication.

[11] Harel, D. Complete Axiomatization of Properties of Recursive Programs. Submitted for publication.

[12] 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, 261-268, Boulder, Col., May 1977.

[13] 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, 249-260, Boulder, Col., May 1977.

[14] Hewitt, C.E. Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot, MIT AI Lab TR-258. 1972.

[15] Hoare, C.A.R. Some Properties of Nondeterministic Computations. The Queen's Univ., Belfast. 1976

[16] Kroeger, F. Logical Rules of Natural Reasoning about Programs. In *Automata, Languages and Programming 3* (ed. Michaelson, S. and R. Milner), 87-98. Edinburgh University Press, 1976.

[17] Manna, Z. Second Order Mathematical Theory of Computation. Proc. 2nd Ann. ACM Symp. on Theory of Computing, 158-168, 1970.

[18] Manna, Z. and A. Pnueli. Axiomatic Approach to Total Correctness of Programs. Acta Informatica, 3, 253-263, 1974.

[19] 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.

[20] Meyer, A.R. Equivalence of DL, DL$^+$ and ADL for Regular Programs with Array Assignments. Unpublished report, MIT. August 1977.

[21] Milner, R.G. An Approach to the Semantics of Parallel Programs. Computer Science Dept., University of Edinburgh, U.K. 1973.

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

[23] deRoever, W.P. Dijkstra's Predicate Transformer, Nondeterminissm, Recursion, and Termination. I.R.I.S.A., Publication Interne No.37. 1976.

[24] Salwicki, A. Formalized Algorithmic Languages. Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys. Vol. 18. No. 5. 1970.

[25] Wand, M. A Characterization of Weakest Preconditions. JCSS 15, 2. 209-212. 1977.

[26] Wang, A. An Axiomatic Basis for Proving Total Correctness of Goto Programs. BIT 16, 88-102. 1976.

[27] Woods, W.A. Augmented Transition Networks for Natural Language Analysis. Report No CS-1 to the NSF, Aiken Computation Laboratory, Harvard University, Cambridge. 1969.