

MIT/LCS/TR-200

Logics of Programs:  
Axiomatics and Descriptive Power

David Harel

*This blank page was inserted to preserve pagination.*

MIT/LCS/TR-200

**Logics of Programs: Axiomatics and Descriptive Power**

by

**David Harel**

**May 1978**

This research was supported in part by the Yad-Avi Rothchild Foundation in Israel, and in part by the National Science Foundation under contract no. MCS76-18461.

**Massachusetts Institute of Technology  
Laboratory for Computer Science**

**Cambridge**

**Massachusetts 02139**

***Keywords:***

arithmetical axiomatization  
arithmetical completeness  
axiom system  
computation tree  
divergence  
dynamic logic  
execution method  
failure  
first-order logic  
guarded commands  
logics of programs  
propositional dynamic logic  
recursive program  
regular program  
relative completeness  
total correctness  
weakest precondition



# **Logics of Programs: Axiomatics and Descriptive Power**

by  
David Harel

Submitted to the Department of Electrical Engineering and Computer Science  
on May 9, 1978, in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy.

## **Abstract.**

This thesis is concerned with the development of mathematical tools for reasoning about computer programs. The approach is to design and investigate the properties of various *dynamic logics* with an emphasis on useful expressive power and adequate proof theory.

First, rigorous definitions of the propositional and first-order dynamic logics are given, with an emphasis on the flexibility obtained by leaving unspecified the class of programs which these logics can discuss. A large portion of the results obtained to date in the investigation of dynamic logic is included and put in proper perspective. Then, a proof theory is developed based upon the idea of axiomatizing the first order dynamic logics relative to *arithmetical universes*. Such axiomatizations are supplied and proved *arithmetically complete* for the regular (flowcharts) and context-free (recursive programs) cases. The notions of *diverging* and *falling* are then introduced, with the aid of which the concept of the *total correctness* of a nondeterministic program is defined and the concept of a *weakest precondition* clarified. A detailed investigation of the properties of diverging and falling is then carried out, including the construction of arithmetically complete axiomatizations of both the regular and context-free logics obtained by supplying dynamic logic with the ability to discuss diverging directly.

Throughout, the presentation stresses the need to be able to *express* interesting properties of programs and to be able to *prove* them when true.

**Thesis Supervisor:** Vaughan R. Pratt, Associate Professor of Computer Science.

## **Acknowledgments.**

The first thanks go to my dear wife Varda who, together with our young daughters Sarit and Hadas, has managed to put up with so much so well, and has provided me with more warmth, love and encouragement than anyone could have wished for.

I am grateful for the unparalleled opportunity to have worked with Professors Albert Meyer and Vaughan Pratt over the past twenty months. Albert's ways of thought, his inspiring and challenging ideas and thorough and critical comments have no doubt had the most profound influence on whatever research capabilities I have developed during my MIT experience. Vaughan's originality and insight were a constant and valuable guide for me, and without his pioneering work on dynamic logic I would still be in search of a thesis topic.

I would like to thank Nachum Dershowitz and Adi Shamir for many hours of stimulating discussions on a variety of topics related to the material in this thesis. I have also greatly benefited in one way or another from talking to, corresponding with, or receiving comments from Edgar W. Dijkstra, Michael Fischer, Carl Hewitt, Jeffery Jaffe, Richard Ladner, Rohit Parikh, Amir Pnueli, Michael Rabin and Karl Winkmann. Thanks to them all.

Albert Meyer and Adi Shamir did a great job as readers, contributing significantly to the clarity and technical correctness of a large portion of the text. Also thanks to Jon Doyle for a voluntary last minute proof reading. A special thanks goes to Gloria Marshall for helping me survive whatever bureaucratic hardships I encountered.

Financial support was kindly provided by a two year grant from the Yad-Avi Rothchild Foundation in Israel through the offices of Bar-Ilan University, and by NSF grant no. MCS76-18461. This document was prepared with the aid of Vaughan Pratt's DOC editing system.

## Table of Contents.

Abstract.	1
Acknowledgments.	2
Table of Contents.	3
0. Introduction.	6
0.1. History.	7
0.2. Synopsis.	8
0.3. Credits.	10
<i>Part I: Binary-Relation Based Logics.</i>	
1. Regular Propositional Dynamic Logic (PDL).	12
1.1. Definitions.	12
1.1.1. Elementary PDL (EPDL).	12
1.1.2. PDL.	14
1.2. Results.	16
1.3. Axiomatization of PDL.	19
2. Regular First-order Dynamic Logic (DL).	22
2.1. Definitions.	22
2.2. Descriptive Power.	27
2.3. Variations.	28
2.3.1. Array Assignment.	29
2.3.2. Random Assignment.	29
2.3.3. Rich Test.	31
2.3.4. Deterministic Dynamic Logic (DDL).	31
2.3.5. R.e. Dynamic Logic.	33
2.4. The Validity Problem for DL.	34

3. Arithmetical Axiomatization. . . . .	36
3.1. The Theorem of Completeness and Arithmetical Universes. . . . .	37
3.2. Axiomatization of DL. . . . .	42
3.3. A Derived Axiomatization of DDL. . . . .	48
3.4. Related Work. . . . .	50
3.4.1. Relative vs. Arithmetical Completeness. . . . .	50
3.4.2. Infinitary Axiomatization. . . . .	53
4. Recursive Programs: Context-free Dynamic Logic (CFDL). . . . .	57
4.1. Definitions. . . . .	57
4.2. Results. . . . .	61
4.3. Axiomatization of CFDL. . . . .	64
4.4. Mutual Recursion. . . . .	68

*Part II: Computation-Tree Based Logics.*

5. Computation Trees, Total Correctness and Weakest Preconditions. . . . .	71
5.1. Motivation. . . . .	72
5.2. Computation Trees, Diverging and Failing. . . . .	73
5.3. Execution Methods and Total Correctness. . . . .	80
5.4. Weakest Preconditions. . . . .	82
5.5. The Guarded Commands Language (GC). . . . .	85
6. The Mathematics of Diverging and Failing I. . . . .	91
6.1. Diverging and Failing in DL. . . . .	91
6.1.1. Expressing $loop_\alpha$ in DL. . . . .	92
6.1.2. Expressing $fail_\alpha$ in DL. . . . .	94
6.2. DL Augmented with $loop_\alpha$ ( $DL^+$ ). . . . .	97
6.2.1. Definitions. . . . .	97
6.2.2. Axiomatization of $DL^+$ . . . . .	99
6.3. A Pattern of Reasoning. . . . .	104
6.4. DL with an Iteration Quantifier (ADL). . . . .	107

7. The Mathematics of Diverging and Failing II. . . . .	110
7.1. Computation Trees for Recursive Programs. . . . .	110
7.2. Diverging and Failing in CFDL. . . . .	117
7.2.1. Expressing $loop_{\omega}$ in CFDL. . . . .	117
7.2.2. Expressing $fail_{\omega}$ in CFDL. . . . .	122
7.3. CFDL Augmented with $loop_{\omega}$ (CFDL <sup>+</sup> ). . . . .	122
7.3.1. Definitions. . . . .	123
7.3.2. Axiomatization of CFDL <sup>+</sup> . . . . .	123
7.4. Language Dependent Diverging and Failing. . . . .	127
8. Conclusion and Directions for Future Work. . . . .	131
Appendix A: Relational Characterization of EPDL. . . . .	133
Appendix B: Example of a Proof of a DL-wff in $P$ . . . . .	136
Appendix C: Example of a Proof of a CFDL-wff in $R$ . . . . .	139
Appendix D: Example of a Proof of a DL <sup>+</sup> -wff in $P^+$ . . . . .	141
Appendix E: Example of a Proof of a CFDL <sup>+</sup> -wff in $R^+$ . . . . .	144
References. . . . .	146
Biographical Note. . . . .	152

## O. Introduction.

At one time or another, every programmer has come across the need to be able to state some property of his program or programs in an unambiguous way. Quite often this property is related in some way to the correctness of the program: "this program sorts its input in ascending order", "this program right-justifies a paragraph of input text" etc. Often it is an undesirable property that is of interest: "this program contains an infinite loop", "this PL/I translation of this Fortran program does not behave exactly as the original" etc. Certainly these statements are not precise and cannot be taken as a basis for a serious discussion about the program in question. Moreover, the need might arise, whether initiated by the programmer himself or by an outsider, to supply some kind of *proof* of the truth of such claims.

In this thesis we take upon ourselves the development of mathematical tools for expressing interesting assertions about programs and for proving those of them which, in a well defined sense, are true. These two concerns, *expressing* and *proving*, will serve as landmarks throughout the thesis. Various formal logics are defined, the motivation for constructing them lying in the kinds of things we would like to be able to *express*; then axiom systems are developed for them, the motivation being rooted in the need to be able to *prove* those things. This, then, explains our title.

We believe that the virtues of research in this area are mainly in providing a sound and rigorous foundational basis upon which reasoning about programs can be carried out. It is not essential, in our opinion, to carry out a proof of the correctness of every program one writes, and certainly not a proof within some formal axiom system. However, it is important to possess the ability of doing so when required. In addition, work in logics of programs provides a theoretical basis for developing computer-aided tools for reasoning about programs, such as interactive verifiers or automatic proof-checkers. We are also of the opinion that, much as a mathematician, when proving a theorem in algebraic topology, benefits from his knowledge of, say, the basics of predicate calculus, an understanding of issues such as those discussed in this thesis results in a subconscious accumulation of important programming knowledge. This knowledge, attainable even at the level of an ordinary programmer, includes understanding the inner workings of such basic programming concepts as sequencing, choice, iteration, recursion, infinite computations etc.

The remainder of this introduction is devoted to a brief historical account of work which influenced the development of the material presented (Section 0.1), a Chapter-by-Chapter summary and description of what is to come (Section 0.2) and a short explanation of the policy adopted, by which some work other than the author's own is also included (Section 0.3).

## 0.1 History.

Early work towards providing mathematical tools for reasoning about programs dates back to Turing [65] and von Neumann [66]. However, it is generally accepted that the first serious attempts solely devoted to that end are those of Floyd [17] and Naur [46] on the *invariant assertion* method for proving the partial correctness of programs, followed by the introduction, by Hoare [27], of an axiom system incorporating that method.

The work we present in this thesis is to a great extent based on Pratt's [52] foundational study of the semantics of Floyd-Hoare logic. (In fact, a preliminary version of [52] in the form of class notes, was written by Pratt in April 1974.) It is in [52] that the "modal logic of programs" (later termed *dynamic logic*, or DL, in [22]) was suggested as a powerful tool, touching off work by Fischer and Ladner [16] on the propositional version, and further work by Harel, Meyer and Pratt [22], Harel and Pratt [25], Pratt [53], Harel [20], [21], Parikh [48], [49], Berman and Paterson [9] and more.

The idea of constructing first-order-like logics for reasoning about programs is not new. A logic quite similar in conception to DL, *algorithmic logic*, has been defined by Salwicki [59] following work of Engeler [15]. Not unlike the situation with DL, Salwicki's original paper stimulated researchers at the University of Warsaw and resulted in extensive study branching off in various directions. Some sample papers are Mirkowska [41], Kreczmar [33], Banachowski [6] and Rasiowa [55]. A survey of their work can be found in [7]. Interestingly, a definition of dynamic logic appears in an appendix of Schwarz [60] and is credited there to Reynolds. However, the idea was not pursued any further there. Also, a very similar logic has been studied for quite a while by Constable, and reported on in [11]. Some comments concerning the relationships holding between DL, algorithmic logic, and Constable's logic appear in [21].

A large amount of related work, which has been of considerable help in developing the material presented, has been published over the years. Some notable examples are

Manna's work in [37] and [38], on the formalization of Floyd's method and related concepts, Cook's [12] relative completeness result for Hoare's axiom system, the work of de Bakker et al [3], [4] and [5] and that of Hitchcock and Park [26] on recursive programs, the completeness results of Harel, Pnueli and Stavi [23] and Corelick [19] for recursive programs, and Dijkstra's [13] logic of total correctness.

## 0.2 Synopsis.

This thesis consists of seven chapters which are organized into two parts. At the end of this section we show some possible self-contained subsets of it which can be read independently.

Part I is concerned with logics which reason about programs based upon their input-output behavior. Here programs (nondeterministic ones in the general case) are viewed as *binary relations* on states, with the intuition that a pair of states is related via a program  $\alpha$  iff starting in the first,  $\alpha$  can terminate in the second. Two primitive notions relevant to this level of description are the one asserting that  $P$  is true in all final states accessible from a given state via the program, and its dual, asserting that there exists such a final state in which  $P$  is true. The idea of *dynamic logic*, due in large to Pratt [52], is to augment a classical "static" logic such as predicate calculus with primitives for expressing these notions, and to use ideas borrowed from Kripke's [34] work on modal logic for defining the semantics of the resulting language.

Chapter 1 provides a definition of PDL, the propositional version of dynamic logic, together with results concerning (a) the decidability of its validity problem, (b) the power obtained by allowing propositional programs to test their environment, and (c) the problem of completely axiomatizing it.

In Chapter 2, the first order version of dynamic logic over *regular* (flowchart) programs, DL, is rigorously defined using the notions of state, universe, and uninterpreted symbols. It is shown that many interesting and well known properties of programs, such as partial correctness and equivalence, can be quite succinctly expressed as formulae of DL. Section 2.3 is aimed at showing that the class of programs allowed in DL is in fact a parameter, and that different classes of programs give rise to different variants of DL. Some open problems concerning the comparative expressive power of these



variations are stated. Section 2.4 contains results which show that validity for DL and some simple sublanguages is extremely hard to decide.

In Chapter 3 we show how an intuitive way in which assertions about programs can be proved is captured formally by allowing the reasoning to be carried out in a first-order language in which, besides any other domain of discourse, the natural numbers and operations on them have their standard interpretations. This is done by introducing the notion of an *arithmetical universe*, and then showing that it is possible to give a concise axiomatization of DL which is complete relative to any such universe. We do not require programs to be written over these universes, but since any universe can be extended to an arithmetical one, this kind of reasoning can always, in principle, be carried out. We show, in Section 3.4, that *arithmetical completeness* is strongly related to Cook's [12] notion of relative completeness, and also discuss the approach of supplying DL with an infinitary, but absolutely complete, axiomatization.

In Chapter 4 we extend the definitions and results of Chapter 3 to the case in which the programs are allowed to be *recursive*. The recursive program construct introduced is simple enough so that a clear analogy between reasoning about iteration and recursion emerges. In particular, the axiomatization, in Section 4.3, of the resulting logic CFDL is far more natural and concise than would have been expected from studying the relevant literature.

Part II is concerned with the two operational notions of *diverging* and *falling* (i.e. entering an "infinite loop" and aborting due to the failing of a test) which are captured naturally by *computation trees*. These trees carry in their leaves the information present in the binary relations of Part I, but also contain information regarding e.g. the presence of divergences and failures. In Chapter 5 we define these new concepts and immediately apply them to the problem of defining a plausible notion of the *total correctness* of a general nondeterministic program. As it turns out, executing a program corresponds to traversing its computation tree, a task for which there are four natural methods, dual to one another. We show that each of these methods gives rise to a different notion of total correctness, and hence to a different notion of the *weakest precondition* which, if true before execution, guarantees total correctness. A detailed analysis is carried out in Sections 5.4 and 5.5 aimed at showing which (if any) of our four notions is the one described informally by Dijkstra [13] and which has been widely adopted for somewhat mysterious reasons.

Chapter 6 is devoted to investigating the mathematical properties of diverging and failing. In particular, it is shown in Section 6.1 that both these notions are expressible in DL, albeit by complicated formulae which have some undesirable properties. In Section 6.2 we augment DL to  $DL^+$  by providing it with the power to express diverging directly, and show that this augmentation gives rise to a surprisingly elegant and natural arithmetically complete axiomatization of the notion of diverging, to be contrasted with the axiomatization obtained by translating this notion into its DL equivalent and then relying on the axiomatization of DL. In Section 6.3 we show that there is a pretty pattern of dualities associated with the construction of arithmetical axiom systems for DL and  $DL^+$ . In Section 6.4 we use the observations inspired by this pattern to obtain a straightforward axiomatization of a related logic, ADL.

Chapter 7 is concerned with supplying results analogous to those of Chapter 6 for the case of recursive programs. Here special methods have to be developed in order to be able to completely axiomatize  $CFDL^+$ , i.e.  $CFDL$  augmented with diverging, and in addition we can only get halfway through showing that  $CFDL$  is powerful enough to express diverging. Consequently, a question which arises is that of whether the results in these sections indicate the existence of some inherent difficulty in reasoning about recursive programs. We cannot supply more than intuition towards answering it. Section 7.1 contains a definition of plausible notions of diverging and failing which do not depend on computation trees and which generalize to other classes of programs too.

As far as reading the thesis is concerned, after reading Chapters 1 and 2 (which are a prerequisite for any other chapter) the reader will have a good understanding of the basics of dynamic logic. He can then read Chapter 5 thus completing a reading aimed at grasping the main definitions for the regular case. Sequences 1,2,3 or 1,2,3,4 confine the reader to dynamic logic (no extensions) but, in addition, provide a treatment of arithmetical completeness for the regular and context-free cases respectively. One might also read 1,2,3,5,6 thus skipping anything to do with recursive programs.

### 0.3 Credits.

The occasion of writing this thesis has provided a good opportunity (and excuse) for preparing a coherent and comprehensive description of the work done recently (mostly by members of the Theory of Computation Group of the Laboratory of Computer Science at

MIT) concerning a new approach towards reasoning about programs, to which the general term *dynamic logic* has been attached. This opportunity has been taken advantage of, and consequently some of the material in the thesis is not due to the author. Any result which is not original with the author is stated with a reference to its originator. Also, we do not supply proofs of results which are not our own, but rather occasionally comment briefly as to the method involved. A consequence is the fact that many results are stated here for the first time and, as of now, no adequate documentation of their proofs is available. We feel, however, that these technicalities are irrelevant when balanced against the virtues of the kind of presentation we have chosen. Following is a quick reference to the notable parts of the thesis which are not original with the author, most of which are included in Chapters 1 and 2.

The ideas upon which the definition of PDL is based are due to V.R. Pratt, and were published, in somewhat different form, in [52]. The definition of PDL in Chapter 1 is due to M.J. Fischer and R.E. Ladner and was published in [16]. The author's own contributions in that chapter are confined to the introduction of EPDL in Section 1.1.1 and its investigation in Appendix A. The material in Chapter 2, also stemming from the ideas of Pratt [52], was developed over a long period jointly by A.R. Meyer, V.R. Pratt and the author (with the exception of Section 2.4 with which the author had little to do). A preliminary version of the rigorous definition of DL presented here was published in [22].

Some of the ideas present in the definition of the computation trees in Section 5.2, in particular the concept of failing, were worked out by the author jointly with V.R. Pratt, and appeared in preliminary form in [25]. The motivation for developing the material in that section was influenced in large by discussions with N. Dershowitz. As noted in the text, the central theorem in Section 6.1.1 is based on a result of Winkmann [71].

Section 7.4 is based upon an idea of A.R. Meyer.

I would like to take this opportunity to express my gratitude to the aforementioned individuals for allowing me to include their own work in this thesis.

## **PART I: Binary-Relation Based Logics.**

### **1. Regular Propositional Dynamic Logic (PDL).**

PDL is the propositional version of dynamic logic, and was defined by M.J. Fischer and R.E. Ladner in [16] "[to] play a role in the logic of programs analogous to the role the propositional calculus plays in the classical first-order logic." They comment, "We have attempted to abstract from [work on logics of programs] the 'pure' logical structure underlying these formal systems. We feel a thorough understanding of this structure is a prerequisite to obtaining a good grasp on the more complicated, albeit more applicable, systems, just as classical propositional logic is fundamental to the understanding of first-order predicate calculus."

We first define an elementary version of PDL (EPDL) aimed at capturing the structure of the interface between programs and formulae, regardless of the kinds of programs involved. We then define PDL essentially as in [16], and state some results concerning PDL and a set of variations  $PDL_i$  for  $i \geq 0$ .

#### **1.1 Definitions.**

##### **1.1.1 Elementary PDL (EPDL).**

EPDL is basically a modal logic with possibly more than one modality. Consequently, the semantics we provide for EPDL are Kripke semantics [34] of modal logic extended to allow many modalities.

##### *Syntax:*

We have two sets of symbols, AF and AP, standing for *atomic formulae* and *atomic programs*. We use  $p, q, \dots$  and  $a, b, \dots$  respectively to denote elements of these two sets

The set of *well-formed formulae* of EPDL (EPDL-wffs) is defined inductively as follows:

- (1) All elements of AF are EPDL-wffs.

- (2) For every  $a$  in  $AP$  and EPDL-wffs  $P$  and  $Q$ ,  
 $(P \vee Q)$ ,  $\neg P$  and  $\langle a \rangle P$  are EPDL-wffs.

We abbreviate  $\neg(\neg P \vee \neg Q)$  to  $P \wedge Q$ ,  $\neg P \vee Q$  to  $P \supset Q$ ,  $(P \supset Q) \wedge (Q \supset P)$  to  $P \equiv Q$ , and  $\neg \langle a \rangle \neg P$  to  $\Box a P$ . We will often omit parentheses, using double spacing when appropriate to prevent ambiguities. The construct  $\langle a \rangle P$  is read "diamond- $a$   $P$ ", and  $\Box a P$  "box- $a$   $P$ ".

*Semantics:*

The central notion in the semantics of EPDL is that of a *universe*  $W$ , which is a nonempty set, each element of which can be thought of as a *state* or *world* in which certain facts are true and others are not. We use  $s, t, \dots$  to denote states. Thus our semantics will have to specify for each EPDL-wff  $P$  and state  $s \in W$ , whether  $P$  is true in  $s$  ( $s$  satisfies  $P$ ) or not. Hence it is plausible to define the meaning of such a formula as the subset of  $W$  consisting precisely of those states which satisfy it. Furthermore, when viewing programs as objects which can "change the state of the world", it is plausible to define the meaning of a program as a *binary relation* on states, including the pair  $(s, t)$  in that relation iff the program in question started in state  $s$  can indeed terminate in state  $t$ . Thus our programs are *nondeterministic*; there can, for a given  $s$ , be more than one  $t$  such that  $(s, t)$  is in that relation.

A *structure*  $S$ , then, is defined as a triple  $(W, \pi, m)$ , where  
 $W$  is a nonempty set,  
 $\pi: AF \rightarrow 2^W$ , and  
 $m: AP \rightarrow 2^{W \times W}$ .

Thus,  $\pi$  and  $m$  provide the meanings for the basic formulae and programs (i.e.  $AF$  and  $AP$ ).

$\pi$  is extended inductively to the set of EPDL-wffs as follows:

$$\begin{aligned}\pi(P \vee Q) &= \pi(P) \cup \pi(Q) = \{s \mid s \in \pi(P) \text{ or } s \in \pi(Q)\}, \\ \pi(\neg P) &= W - \pi(P) = \{s \mid s \notin \pi(P)\}, \\ \pi(\langle a \rangle P) &= \{s \mid (\exists t)((s, t) \in m(a) \text{ and } t \in \pi(P))\}.\end{aligned}$$

Denoting  $s \in \pi(P)$  by  $s \models P$  and  $(s, t) \in m(a)$  by  $s \text{ sat } a$  and adopting free usage of conventional logical symbols in our discussions, we may write for fixed  $S$ :

$$s \Vdash \langle a \rangle P \text{ iff } \exists t (s a t \wedge t \Vdash P)$$

reading: "diamond- $a$   $P$  is true in state  $s$  iff there exists a state reachable from  $s$  via  $a$ , which satisfies  $P$ ". One may then verify that for  $\langle a \rangle P$  (defined as an abbreviation to  $\neg \langle a \rangle \neg P$ ) we have

$$s \Vdash \langle a \rangle P \text{ iff } \forall t (s a t \supset t \Vdash P)$$

reading: "box- $a$   $P$  is true in state  $s$  iff every state reachable from  $s$  via  $a$  satisfies  $P$ ".

Given a structure  $S = (W, \pi, m)$  we say that an EPDL-wff  $P$  is  $S$ -valid (and write  $\models_S P$ ) if for every  $s \in W$  we have  $s \Vdash P$ . We say  $P$  is valid (and write  $\models P$ ) if it is  $S$ -valid for every structure  $S$ .  $P$  is said to be  $S$ -satisfiable if for some  $s \in W$  we have  $s \Vdash P$ , and *satisfiable* if it is  $S$ -satisfiable for some  $S$ . The following are examples of valid EPDL-wffs:

$$\begin{aligned} (\langle a \rangle p \wedge \langle a \rangle \text{true}) &\supset \langle a \rangle p, & \text{where true abbreviates } \bigvee q, \\ \langle a \rangle (p \wedge q) &\supset (\langle a \rangle p \wedge \langle a \rangle q), \\ \langle a \rangle (p \vee q) &= (\langle a \rangle p \vee \langle a \rangle q). \end{aligned}$$

The first example states essentially that if wherever you go  $p$  holds and if furthermore you can go somewhere, then you can go somewhere where  $p$  holds.

At this point we refer the reader to Appendix A where we define an interesting *relational algebra* which employs only two operations on relations: the conventional composition operator  $\circ$  and the new unary operator *minus*  $(-)$  defined as

$$-e = \{(s, s) \mid \forall t ((s, t) \notin e)\}.$$

We show there how to embed EPDL in this algebra, thus capturing EPDL in a pure relational format, and point to some questions which seem to justify further investigation of this direction.

### 1.1.2 PDL.

In the propositional dynamic logic we now describe, the set of programs is taken to be the set of *regular expressions* over  $\Lambda P$ . This facilitates reasoning about "structured" programs built up from atomic, uninterpreted, program letters.

**Syntax:**

Here too we have the two sets of symbols AF and AP, and in addition we require that AP contain one special element, denoted by  $\emptyset$ , which corresponds to the empty program.

The set  $R$  of *regular programs* is defined inductively as follows:

- (1) All elements of AP are in  $R$ ,
- (2) For all  $\alpha$  and  $\beta$  in  $R$ ,  $(\alpha;\beta)$ ,  $(\alpha\cup\beta)$  and  $\alpha^*$  are in  $R$ .

The set of *well-formed formulae* of PDL (PDL-wffs) is defined inductively similarly to EPDL:

- (1) All elements of AF are PDL-wffs,
- (2) For every  $\alpha$  in  $R$  and PDL-wffs  $P$  and  $Q$ ,  
( $P\vee Q$ ),  $\neg P$  and  $\langle\alpha\rangle P$  are PDL-wffs.

We abbreviate as in Section 1.1.1.

**Semantics:**

Here too we have the notion of a *structure*  $S = (W, \pi, m)$ . However, we are now obliged to extend  $m$  to the class of programs  $R$ . This is done as follows:

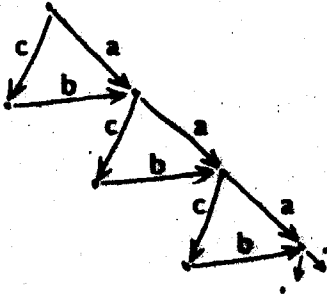
$$\begin{aligned} m(\emptyset) &= \emptyset, \\ m(\alpha;\beta) &= m(\alpha) \circ m(\beta) = \{(s,t) \mid (\exists u)((s,u) \in m(\alpha) \text{ and } (u,t) \in m(\beta))\} \\ m(\alpha\cup\beta) &= m(\alpha) \cup m(\beta) = \{(s,t) \mid (s,t) \in m(\alpha) \text{ or } (s,t) \in m(\beta)\} \\ m(\alpha^*) &= (m(\alpha))^* = \{(s,t) \mid (\exists i \geq 0)(\exists s_0, s_1, \dots, s_i) \\ &\quad (s_0 = s \text{ and } s_i = t \text{ and } (\forall j > 0)(s_{j-1}, s_j) \in m(\alpha))\} \end{aligned}$$

Here the double usages of  $\cup$  and  $*$  on both sides of the equation represent operations in the formal language we are defining and operations on binary relations respectively; in the latter  $\cup$  is union and  $*$  is reflexive and transitive closure. Thus, our programs are literally the regular expressions over the alphabet AP, with  $\emptyset$ ,  $\alpha;\beta$ ,  $\alpha\cup\beta$ , and  $\alpha^*$  meaning respectively "the empty program", "do  $\alpha$  followed by  $\beta$ ", "do either  $\alpha$  or  $\beta$  the choice being nondeterministic", and "do  $\alpha$  any (nonnegative) number of times the choice being nondeterministic". Here "doing  $\alpha$  0 times" is like "doing nothing".  $\pi$  is extended inductively to the set of PDL-wffs as in EPDL, and the definitions of validity and satisfiability are the same too. The following are examples of valid PDL-wffs:

$$\begin{aligned} \langle a \cup b \rangle (p \wedge q) &\supset ((\langle a \rangle p \wedge \langle a \rangle q) \vee (\langle b \rangle p \wedge \langle b \rangle q)), \\ [a^*; a^*] p &\equiv [a^*] p, \\ ([a; a]^* p \wedge [a; (a; a)^*] \neg p) &\equiv (p \wedge [a^*] (p \supset [a] \neg p \wedge (\neg p) \supset [a] p)). \end{aligned}$$

The last of these (due to A.R. Meyer) asserts the equivalence of two ways of stating that  $p$  and  $\neg p$  hold alternatively along arbitrary  $a$ -paths.

Taking *false* to abbreviate  $p \wedge \neg p$ , the following are examples of  $S$ -valid formulae where the structure  $S$  is described by the self explanatory diagram



$$\begin{aligned} \langle a \cup b \rangle [a^*] \langle a; (a \cup c) \rangle (\langle b \rangle \text{true} \wedge [a] \text{false}), \\ \langle a \cup b \rangle [a^*] \langle a; (a \cup c) \rangle (\langle b \rangle \text{false} \wedge \langle a \rangle \text{true}). \end{aligned}$$

## 1.2 Results.

First we state some straightforward consequences of our definitions and provide proofs of some representative cases.

**Lemma 1.1:** For every PDL-wff  $P$  and  $\alpha, \beta \in R$ , the following are valid:

- (a)  $[ \alpha; \beta ] P \equiv [ \alpha ] [ \beta ] P$ ,
- (b)  $[ \alpha \cup \beta ] P \equiv ([ \alpha ] P \wedge [ \beta ] P)$ .

*Proof:* We prove (a).  $[ \alpha; \beta ] P$  iff  $\forall i (s \alpha; \beta \supset i P)$  iff  $\forall i (\exists u (s \alpha \wedge u \beta) \supset i P)$  iff  $\forall i, u ((s \alpha \wedge u \beta) \supset i P)$  iff  $\forall i, u (s \alpha \supset (u \beta \supset i P))$  iff  $\forall u (s \alpha \supset \forall i (u \beta \supset i P))$  iff  $\forall u (s \alpha \supset u [ \beta ] P)$  iff  $[ \alpha ] [ \beta ] P$ . ■



**Lemma 1.2:** For every PDL-wff  $P$  and  $\alpha \in R$ ,  $s \models [\alpha^*]P$  iff for every  $n \geq 0$  we have  $s \models [\alpha^n]P$ , where  $\alpha^0$  is true? and  $\alpha^{n+1}$  is  $\alpha; \alpha^n$ .

*Proof:*  $s \models [\alpha^*]P$  iff  $\forall t (s \alpha^* t \supset t \models P)$  iff  $\forall t ((\exists n \exists s_1 \dots s_n) (s \alpha s_1 \wedge \dots \wedge s_{n-1} \alpha s_n \wedge s_n = t) \supset t \models P)$  iff  $\forall n \forall t (s \alpha^n t \supset t \models P)$  iff for every  $n \geq 0$ ,  $s \models [\alpha^n]P$ . ■

**Lemma 1.3:** For every  $\alpha \in R$  and PDL-wffs  $P$  and  $Q$  the following are valid:

- (a)  $[\alpha](P \wedge Q) \equiv ([\alpha]P \wedge [\alpha]Q)$ ,
- (b)  $[\alpha](P \supset Q) \supset ([\alpha]P \supset [\alpha]Q)$ ,
- (c)  $\langle \alpha \rangle (P \vee Q) \equiv (\langle \alpha \rangle P \vee \langle \alpha \rangle Q)$ ,
- (d)  $\langle \alpha \rangle (P \wedge Q) \supset (\langle \alpha \rangle P \wedge \langle \alpha \rangle Q)$ .

*Proof:* We prove (a).  $s \models [\alpha](P \wedge Q)$  iff  $\forall t (s \alpha t \supset t \models (P \wedge Q))$  iff  $\forall t (s \alpha t \supset (t \models P \wedge t \models Q))$  iff  $(\forall t (s \alpha t \supset t \models P) \wedge \forall t (s \alpha t \supset t \models Q))$  iff  $s \models ([\alpha]P \wedge [\alpha]Q)$ . ■

Note that a trivial counter-example to the other direction of both (b) and (d) is the structure with two states  $s$  and  $t$  in which  $P$  is true only in  $s$  and  $Q$  only in  $t$ , and in which we have both  $s \alpha s$  and  $s \alpha t$ .

**Theorem 1.4 (Fischer and Ladner [16]):** The validity problem for PDL is decidable.

This result is obtained by establishing a "finite model theorem" for PDL, stating that a PDL-wff is satisfiable iff it is  $S$ -satisfiable for some structure  $S$  in which the universe  $W$  is finite and in fact bounded by an exponential in the size of the wff. The following theorem essentially establishes an upper bound on this decision method.

**Theorem 1.5 (Fischer and Ladner [16]):** Satisfiability in PDL can be decided in nondeterministic time  $c^n$  for some constant  $c$ , where  $n$  is the length of the formula tested.

Pratt [53] has recently developed a decision procedure for PDL, based on the tableau method, which, in many naturally arising cases, is more efficient than the one implicit in the proof of Theorem 1.5 in [16].

**Theorem 1.6 (Fischer and Ladner [16]):** There is a constant  $c > 1$  such that satisfiability in PDL cannot be decided in deterministic time  $c^{n / \log n}$ .

This lower bound is proved by showing how to simulate the computation of an alternating Turing machine with a PDL-wff.

The following results are concerned with a variation of PDL in which programs are allowed to test the truth of certain formulas, implying abortion if the test produces a positive answer and abortion if not.

For the purpose of the rest of this section we let  $PDL_0$  stand for PDL. Now, for any  $i \geq 1$  define  $PDL_i$  inductively as an extension of  $PDL_{i-1}$  by adding to the definition of the set of programs  $R$  the clause

(3) For any  $PDL_{i-1}$ -wff  $P$ ,  $P?$  is in  $R$ ,

and to the definition of the extension of  $m$  to  $R$ , the clause

$$m(P?) = \{(s, s) \mid s \in \pi(P)\}.$$

Thus, for example  $\exists x; \langle a; (P?) \rangle$  is a  $PDL_1$ -wff. Define  $PDL_{\infty} = \bigcup_{i \geq 0} PDL_i$ .

**Lemma 1.7:** For every  $PDL_{\infty}$ -wffs  $P$  and  $Q$ ,  $\langle P? \rangle Q = P \Rightarrow Q$  is a valid  $PDL_{\infty}$ -wff.

*Proof:* Straightforward from the definitions. ■

**Theorem 1.8 (Fischer and Laderer [15]):** Satisfiability in  $PDL_{\infty}$  is decidable in nondeterministic time  $c^n$  for some constant  $c$ .

The following are some well known programming constructs and their translation into PDL with tests:

if $P$ then $\alpha$ else $\beta$	$(P?; \alpha) \cup (-P?; \beta)$ ,
while $P$ do $\alpha$	$(P?; \alpha)^*$ ; $-P?$ ,
IF $P \rightarrow \alpha \parallel Q \rightarrow \beta$ FI	$(P?; \alpha) \cup (Q?; \beta)$ ,
DO $P \rightarrow \alpha \parallel Q \rightarrow \beta$ OD	$((P?; \alpha) \cup (Q?; \beta))^*$ ; $(-P \wedge -Q)?$ .

Note that there is an interesting logic "halfway" between  $PDL_0$  and  $PDL_1$ , namely that in which clause (3) above is taken to be

(3) For any  $p$  in  $AF$ ,  $p?$  is in  $R$ .

Thus, we allow only testing of propositional letters from  $AF$ . Denote this variant of PDL by  $PDL_{0.5}$ .

*Theorem 1.9* (Berman and Paterson [9]): There exists a  $PDL_{0.5}$ -wff  $P$ , such that there is no  $PDL_0$ -wff  $Q$  such that  $\models(P=Q)$  (where  $P=Q$  is to be viewed as a  $PDL_{0.5}$ -wff).

*Theorem 1.10* (Berman [8]): For any  $i \geq 0$ , there exists a  $PDL_{i+1}$ -wff  $P$ , such that there is no  $PDL_i$ -wff  $Q$  such that  $\models(P=Q)$  (where  $P=Q$  is to be viewed as a  $PDL_{i+1}$ -wff).

Informally, these results mean that each "level of testing" supplies increasingly more expressive power, or in other words,

$$PDL_0 < PDL_{0.5}, \quad \text{and} \\ (\forall i \geq 0)(PDL_i < PDL_{i+1}),$$

the second, say, reading "for every  $i$ ,  $PDL_{i+1}$  is strictly more expressive than  $PDL_i$ ".

Theorem 1.9 (and similarly 1.10) is proved by a subtle argument involving the construction of two families of structures  $S_j$  and  $S'_j$  for every  $j \geq 0$ , and the exhibition of a  $PDL_{0.5}$ -wff  $P$  which can "distinguish" between  $S_j$  and  $S'_j$  for all  $j$ . One can then show that corresponding to any  $PDL_0$ -wff  $Q$  there exists an integer  $j(Q) > 0$  such that  $Q$  cannot distinguish between  $S_{j(Q)}$  and  $S'_{j(Q)}$ .

Berman [8] has also shown that  $PDL_{0.5} < PDL_1$ .

### 1.3 Axiomatization of PDL.

A problem left open in [16] was that of finding a natural complete axiom system for PDL. Consider the following axiom system  $X$ :

*Axioms:*

- (1) All tautologies of propositional calculus,

- (2)  $[\alpha](P \supset Q) \supset ([\alpha]P \supset [\alpha]Q)$ ,  
 (3)  $[\alpha]P$ ,  
 (4)  $[\alpha;\beta]P \equiv [\alpha][\beta]P$ ,  
 (5)  $[\alpha\cup\beta]P \equiv ([\alpha]P \wedge [\beta]P)$ ,  
 (6)  $[\alpha^*]P \equiv (P \wedge [\alpha][\alpha^*]P)$ ,  
 (7)  $[\alpha^*](P \supset [\alpha]P) \supset (P \supset [\alpha^*]P)$ ,

**Inference Rules:**

$$(8) \frac{P, P \supset Q}{Q}$$

$$(9) \frac{P}{[\alpha]P}$$

If  $PDL_{\alpha\cup\beta}$  is considered then add the axiom

$$(10) [\alpha\cup\beta]P \equiv Q \supset P,$$

and call the augmented system  $X'$ .

Provability in  $X$  or  $X'$  is defined in the standard way;  $P$  is *provable* (written  $\vdash_X P$ ) if there exists a finite sequence of PDL-wffs such that each is an instance of one of the axioms or is obtained from previous formulas by one of the rules of inference. A version of this system had been conjectured by us for quite a while to be complete, but final confirmation of this fact came recently, independently, in Parikh [40], Pratt [53], Segerberg [61] and Gabbay [10].

**Theorem 1.11** (Parikh, Pratt, Segerberg and Gabbay): For any PDL-wff (resp.  $PDL_{\alpha\cup\beta}$ -wff)  $P$ ,  $\vdash P$  iff  $\vdash_X P$  (resp.  $\vdash_{X'} P$ ).

As an example of a proof in  $X'$ , serving in addition to familiarize the reader with the notions of PDL, we sketch the proof of the validity of the following  $PDL_1$ -wff:

$$[(\langle a \rangle \text{true?}; a)^*]p \supset [a^*]p.$$

Abbreviating  $(\langle a \rangle \text{true?}; a)$  to  $\beta$  and  $[\beta^*]p$  to  $Q$ , we state the main points in the proof omitting reference to (1) and (8). The reader is urged to convince himself that each step can be rigorously justified in  $X$ .

- |     |  |                    |
|-----|--|--------------------|
| 1.  | $Q \supset (Q \vee \text{false}),$                             |                    |
| 2.  | $[a]Q \supset [a](Q \vee \text{false}),$                       | line 1, (9), (2),  |
| 3.  | $[a]\text{false} \supset [a](Q \vee \text{false}),$            | same as line 2,    |
| 4.  | $([a]\text{false} \vee [a]Q) \supset [a](\text{false} \vee Q)$ | lines 2,3,         |
| 5.  | $(\langle a \rangle \text{true} \supset [a]Q) \supset [a]Q,$   |                    |
| 6.  | $[\langle a \rangle \text{true?}; a]Q \supset [a]Q,$           | line 5, (10),      |
| 7.  | $[\beta]Q \supset [a]Q,$                                       | (4), line 6,       |
| 8.  | $Q \supset [\beta]Q,$  | (6),               |
| 9.  | $Q \supset [a]Q,$  | lines 7,8,         |
| 10. | $[a^*](Q \supset [a]Q),$                                       | (9), line 9,       |
| 11. | $Q \supset [a^*]Q,$  | (7), line 10,      |
| 12. | $Q \supset p,$   | (6),               |
| 13. | $[a^*]Q \supset [a^*]p,$                                       | line 12, (9), (2), |
| 14. | $Q \supset [a^*]p.$  | lines 11,13.       |

## 2. Regular First-order Dynamic Logic (DL).

In this chapter we define a first order logic based upon ideas from Pratt [52] further developed in [22]. The logic, *first order dynamic logic*, or **DL** for short, is designed to reason about "real" regular programs; i.e. the equivalent of nondeterministic flowcharts or recursion-free loop programs. The sense in which the programs are real is in that they employ the conventional notions of changing the value of variables by *assigning* to them and *testing* the value of expressions. Programs in DL are no longer combinations of atomic program symbols, and program-free formulae are no longer propositional.

After defining DL we elaborate on the kinds of facts expressible in it. Section 2.3 contains some extensions of and restrictions upon the class of programs allowed in DL, viewing all the resulting logics as variations of DL. Section 2.4 contains results concerning the question of how hard it is to decide the validity of certain kinds of formulae of DL.

### 2.1 Definitions.

#### *Syntax:*

We are given a set of *function symbols* and a set of *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  $p, q, \dots$  and  $k$ -ary function symbols for  $k > 0$  by  $f, g, \dots$ . Zeroary function symbols are denoted by  $z, x, y, \dots$  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.

We define by simultaneous induction the set  $RG$  of first-order regular programs and the set of DL-wffs:

- (1) For any variable  $x$  and term  $e$ ,  $x \leftarrow e$  is in  $RG$ ,
- (2) For any program-free (see below) DL-wff  $P$ ,  $P?$  is in  $RG$ ,
- (3) For any  $\alpha$  and  $\beta$  in  $RG$ ,  $(\alpha; \beta)$ ,  $(\alpha \cup \beta)$  and  $\alpha^*$  are in  $RG$ ,
- (4) Any atomic formula is a DL-wff,
- (5) For any DL-wffs  $P$  and  $Q$ ,  $\alpha$  in  $RG$  and variable  $x$ ,  
 $\neg P$ ,  $(P \vee Q)$ ,  $\exists x P$  and  $\langle \alpha \rangle P$  are DL-wffs.

A DL-wff which contains no occurrence of a program of  $RG$  is called *program free* or simply a *first order formula*. Programs of the form indicated in (1) and (2) are called respectively (simple) *assignments* and (simple) *tests*. We use  $\wedge$ ,  $\supset$ ,  $=$  and  $[\alpha]$  for abbreviations as in the previous chapter, and in addition abbreviate  $\neg \exists x \neg P$  to  $\forall x P$ .

(Remark: As will be seen in Section 2.3, the particular class of programs allowed in DL-wffs can be viewed as being a parameter. Different classes give rise to different variations. Even within the particular class of regular programs the set of tests can be allowed to vary; it can be the set of quantifier-free tests or, inductively, the set of question-marked DL-wffs. Various kinds of assignments are also possible. We stress these facts here, even before completing the definition of DL, so that the reader does not associate any particular class of programs with the generic term *dynamic logic*.)

#### Semantics:

The semantics of DL is based on the concept of a state. The difference however, is that we are now concerned with specific atomic programs and specific atomic formulae.

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  $D_J$ .

Observe that the way states are defined no distinction is made between what are normally called variables and constants. These, however, will be defined below for simple universes.

We denote by  $\Gamma$  the collection of all possible states and call it the *grand universe*. Our semantics will assign to a program  $\alpha$  a binary relation  $m(\alpha)$  over  $\Gamma$ , and to a formula  $P$  a subset of  $\Gamma$  consisting of those states which satisfy  $P$ . In the sequel however, we will be interested in special subsets of  $\Gamma$  namely 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  $J \in U$  and for every function  $F$  (resp. predicate  $P$ ) over  $D$  there exists  $\mathcal{J} \in U$  such that  $J$  and  $\mathcal{J}$  differ at most in the value of  $f$  (resp.  $p$ ) which in  $\mathcal{J}$  is  $F$  (resp.  $P$ ).

*Notation:* For any function  $G: A \rightarrow B$ , arbitrary element  $e$ , and  $a \in A$ , we define  $[e / a]G$  to be the function with domain  $A$  and range  $B \cup \{e\}$  giving the same values at points in  $A - \{a\}$  as  $G$ , and such that  $G(a) = e$ . Thus, the situation described above for uninterpreted  $f$  is simply  $\mathcal{J} = [F / f]J$ .

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.

The value of a term  $e = f(e_1, \dots, e_k)$  in a state  $J$  is defined inductively following Tarski [64], by

$$e_j = f_j(e_{1j}, \dots, e_{kj}).$$

We now define by simultaneous induction the binary relation over  $\Gamma$  corresponding to a program  $\alpha$  of RG, and those states  $J$  in  $\Gamma$  which satisfy a DL-wff  $P$ . The relation will be denoted by  $m(\alpha)$  and for the latter we write  $J \models P$ .  $(J, \mathcal{J})$  being an element of  $m(\alpha)$  can be thought of as representing the fact that there exists a *computation sequence* (or *path*) of  $\alpha$  starting in state  $J$  and terminating in  $\mathcal{J}$ . Thus,  $J \models \alpha \models P$  will be seen to be making an assertion about *all* terminating computations of  $\alpha$  starting in state  $J$ ; namely the



assertion that the final states of these computations satisfy  $P$ . Similarly,  $J \models \langle \alpha \rangle P$  asserts the *existence* of a terminating computation of  $\alpha$  starting in state  $J$  and ending in a state satisfying  $P$ .

- (1') For any variable  $x$  and term  $e$ ,  
 $m(x \leftarrow e) = \{(J, J) \mid J = [e_J / x]J\}$ ,
- (2') for any program-free DL-wff  $P$ ,  
 $m(P?) = \{(J, J) \mid J \models P\}$ ,
- (3') For any  $\alpha$  and  $\beta$  in RC,  
 $m(\alpha; \beta) = m(\alpha) \circ m(\beta)$ ,  
 $m(\alpha \cup \beta) = m(\alpha) \cup m(\beta)$ ,  
 $m(\alpha^*) = (m(\alpha))^*$ ,  
 (see Section 1.1 for further specification)
- (4') For an atomic formula  $p(e_1, \dots, e_k)$ ,  
 $J \models p(e_1, \dots, e_k)$  whenever  $p_J(e_{1J}, \dots, e_{kJ})$  is true,
- (5') For any DL-wffs  $P$  and  $Q$ ,  $\alpha$  in RC and variable  $x$ ,  
 $J \models \neg P$  iff it is not the case that  $J \models P$ ,  
 $J \models (P \vee Q)$  iff either  $J \models P$  or  $J \models Q$ ,  
 $J \models \exists x P$  iff there exists an element  $d$  in  $D_J$  such that  $[d / x]J \models P$ ,  
 $J \models \langle \alpha \rangle P$  iff there exists a state  $J'$  such that  $(J, J') \in m(\alpha)$  and  $J' \models P$ .

Note that the only kinds of formulae whose truth in state  $J$  depends possibly upon states other than  $J$  are those containing subformulae of the form  $\exists x P$  and  $\langle \alpha \rangle P$ .

In most of this thesis we will primarily be interested in investigating the truth of DL-wffs in a given simple universe  $U$ . However, one can see that for some  $J \in U$  and some assignment  $x \leftarrow e$  the unique state  $J'$  such that  $(J, J') \in m(x \leftarrow e)$ , i.e. the state  $[e_J / x]J$ , 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  $x \leftarrow e$ ) and the quantified variables (e.g.  $x$  in  $\exists x P$ ) are uninterpreted. Thus, for  $J \in U$  and for any DL-wff  $P$  the truth of  $J$  in  $P$  can be seen to depend only on states in  $U$ .

We use abbreviations as in Chapter 1, and thus will write  $J\alpha J$  for  $(J, J) \in m(\alpha)$ , and for  $[\alpha]$ , which stands for  $\neg\langle\alpha\rangle\neg$ , we have again

$$J\vdash[\alpha]P \text{ iff } \forall J(J\alpha J \supset J\vdash P).$$

Given a universe  $U$  we say that a DL-wff  $P$  is  $U$ -valid ( $\vdash_U P$ ) if for every  $J \in U$  we have  $J\vdash P$ . We say  $P$  is valid ( $\vdash P$ ) if it is  $U$ -valid for every universe  $U$  in which, in line with the above convention, the assigned and quantified variables of  $P$  are uninterpreted.

The following are examples of valid DL-wffs:

$$\begin{aligned} &[(x=z \wedge y=u)?; (x \leftarrow f(x) \cup y \leftarrow f(y))](x=z \vee y=u), \\ &x=y \supset [(x \leftarrow f(f(x)))^*](y \leftarrow f(y))^* \supset x=y, \\ &x=y \supset [(x \leftarrow f(x))^*](p(x) \supset (x=y \vee \langle y \leftarrow f(y) \rangle^* \supset p(y))). \end{aligned}$$

The first asserts that at most one of the components of  $U$  is executed. The second states that the process of repeatedly applying a function composed with itself is a special case of that of repeatedly applying it. The third asserts essentially that the process of achieving a property of  $x$  by repeatedly applying  $f$  can be simulated in  $y$ .

Denote by  $N$  the simple universe of *pure arithmetic*; i.e. the domain  $D$  is the set of natural numbers and  $+$ ,  $'$  and  $0$  are fixed with their standard interpretations. We freely use standard arithmetical abbreviations such as  $\geq$ ,  $\text{gcd}$  etc. (Whenever, in the context of the natural numbers, we use the symbol  $-$ , it is to be understood to stand for the so called "minus" operation, i.e.  $x-y$  is the difference between  $x$  and  $y$  if  $x \geq y$ , and  $0$  otherwise. Also, we abbreviate  $x=x$  to *true* and  $\neg \text{true}$  to *false*.)

The following are  $N$ -valid DL-wffs:

$$\begin{aligned} &\langle (x \leftarrow x-1)^* \rangle x=0, \\ &y>0 \vee \langle y=0 \rangle \text{true} \\ &[(x=x' \wedge y=y' \wedge x'y>0)?](x \neq y?; \\ &\quad (x>y?; x \leftarrow x-y \cup x < y?; y \leftarrow y-x))^*; x=y? \supset x=\text{gcd}(x', y'). \end{aligned}$$

The last example asserts that the program inside the diamond, under the assumption that its two inputs are positive integers, terminates and computes the *gcd* of these inputs.

This program can be written in more popular terms as:

```

while  $x \neq y$  do
  if  $x > y$  then  $x \leftarrow x - y$ 
  else  $y \leftarrow y - x$ 
end.

```

We adopt the standard definition of a *free occurrence* of a variable  $x$  in a first order formula  $Q$  to be an occurrence of  $x$  which is not in any subformulae of the form  $\exists x P$ . Any other occurrence is called *bound*. Also, we define  $Q_x^e$  for variable  $x$  and term  $e$  to be the formula which is obtained from  $Q$  by uniformly renaming all bound variables of  $Q$  which appear in  $e$  and replacing all free occurrences of  $x$  by  $e$ .

*Lemma 2.1:* For every assignment  $x \leftarrow e$ , and first-order formula  $Q$ ,  
we have  $\models (x \leftarrow e)Q = Q_x^e$ .

## 2.2 Descriptive Power.

One of the virtues of logics such as DL is the fact that they provide a general framework in which it is possible to express a wide variety of concepts and notions, for each of which one would otherwise have to invent a special notation. The advantages of this uniformity are by no means only notational; elementary results and analogies are much more obscure and harder to come by when these concepts are developed under separate cover. This argument is implicit in Sections 3.3 and 6.3 as well as in [21].

The examples in Section 2.1 illustrate some of the facts expressible in DL, the first set of valid ones being true for "uninterpreted" reasons and the others for reasons stemming from properties of the specific universe involved, in this case that of arithmetic. Indeed, when people reason informally about real programs, they generally have in mind a particular interpretation for the symbols appearing in the program. Consequently, we will be more interested in this thesis in providing adequate tools for "domain dependent" reasoning; e.g. for proving the  $U$ -validity of formulae for some universe  $U$ .

Although we wish to stress the fact that one can write complex DL-wffs (e.g. alternations of boxes and diamonds of arbitrary length are certainly permitted), we point to some particular elementary properties of programs and show how to express them, with relatively simple formulae, in DL given a universe  $U$ .

- *Partial correctness of  $\alpha$  wrt P and Q* (Hoare's [27])  $P\{\alpha\}Q$ :  $\vdash_{\perp}(P \supset [\alpha]Q)$ ,
- *Existence of a Q-terminating path of  $\alpha$* :  $\vdash_{\perp}\langle \alpha \rangle Q$ ,
- *Existence of a Q-terminating path of  $\alpha$  under the assumption P*:  $\vdash_{\perp}(P \supset \langle \alpha \rangle Q)$ ,  
(This turns out to be the *total correctness* of  $\alpha$  when  $\alpha$  is deterministic, and has been denoted by a variety of notations, see [21]. Also see Section 2.3.4 and Chapter 5 of this thesis.)

For any  $\alpha \in RC$ , define  $var(\alpha)$  as a finite vector consisting, in some fixed standard order, of all variables appearing to the left of the assignment symbol  $\leftarrow$  in  $\alpha$ .

- *Equivalence of  $\alpha$  and  $\beta$* :  $\vdash_{\perp} \forall Z' (\langle \alpha \rangle Z = Z' \equiv \langle \beta \rangle Z = Z')$ , where  $Z = var(\alpha) = var(\beta)$ , and  $Z'$  is a vector of the same length as  $Z$  whose components are distinct variables not in  $var(\alpha)$ .
- *Determinacy of  $\alpha$  (all terminating paths have a common final state)*:  $\vdash_{\perp} \forall Z' (\langle \alpha \rangle Z = Z' \supset [\alpha] Z = Z')$ , with  $Z$  and  $Z'$  as above.

### 2.3 Variations.

Regular programs of the kind we have employed are by no means mandatory. On the contrary, the reader and any potential user of DL is encouraged to use the basic concepts (as portrayed say in EPDL) freely with his favorite programming language. We ourselves prefer to work with regular (and later with context-free) expressions over assignments and first-order tests. In this section however, we provide a number of possible variations of this set of programs.

We are about to introduce various logics and give them names, and we would like to be able to compare their expressive power. If it is the case then, that for two such logics  $A$  and  $B$ , the wffs of  $A$  are a subset of those of  $B$ , we will denote by  $A < B$  the assertion that there exists a  $B$ -wff  $P$  such that for no  $A$ -wff  $Q$  is it the case that  $P \equiv Q$  is a valid  $B$ -wff.

### 2.3.1 Array Assignment.

An array-assignment is a basic program which can change the value of a function symbol at a specific point. This is done by writing  $f(z) \leftarrow e$  where  $f$ ,  $z$  and  $e$  are respectively, a  $k$ -ary function symbol, a  $k$ -tuple of variables, and a term. We restrict ourselves for simplicity to the case where  $k=1$ .

To obtain this new language, which we call *array-DL*, the following clauses are added to the definitions of the syntax and semantics of DL respectively:

(1a) For any unary function symbol  $f$ , variable  $x$  and term  $e$ ,  
 $f(x) \leftarrow e$  is in RC,

(1a') For any unary function symbol  $f$ , variable  $x$  and term  $e$ ,  
 $m(f(x) \leftarrow e) = \{(J, [F / f]J) \mid F = [e_j / x_j] f_j\}$ .

Note that although a program with array assignments can change the value of  $f$  at unboundedly many points (e.g. as might be the case with the program  $(x \leftarrow g(x); f(x) \leftarrow y)^*$ ), it cannot in general change the "entire" value of  $f$  as in a *second order* assignment of the form  $f \leftarrow g$ , which, although constituting another plausible variation, is not allowed here. We extend our convention of Section 2.1 to require that in the context of a given universe  $U$  we allow array assignments of the form  $f(x) \leftarrow e$  only if  $f$  is uninterpreted in  $U$ .

*Open Problem:* Is  $DL < \text{array-DL}$ ?

Answering this question in the affirmative would involve exhibiting an array-DL-wff  $P$ , and showing that for no DL-wff  $Q$  do we have  $\models (P \equiv Q)$ . Certainly, the obvious fact that certain programs can be written easily and succinctly using array assignments will not be affected by an answer to this question; it is strictly a question about the power of expression of a formal logic for reasoning about these programs.

### 2.3.2 Random Assignment.

A random-assignment is a basic program which in a state  $J$  can change the value of a variable  $x$  nondeterministically to any element of the domain  $D_J$ . Strictly speaking

however, this type of assignment is appropriate (and of use) only when  $x$  is uninterpreted, in which case every element of  $D_j$  is indeed a possible value of  $x$ .

The following clauses are then added to the appropriate places in the definition of DL to obtain *random-DL*:

(1b) For any variable  $x$ ,  $x \leftarrow ?$  is in RC,

(1b') For any variable  $x$ ,  $m(x \leftarrow ?) = \{(j, j) \mid j = [x_j / x]\}$ .

Thus,  $x \leftarrow ?$  when started in  $j$ , can terminate in any state in which only the value of  $x$  has been changed.

*Lemma 2.2:* For any universe  $U$ , uninterpreted variable  $x$  and DL-wff  $P$ , we have  $\models (\exists x P = \langle x \leftarrow ? \rangle P)$  and  $\models (\forall x P = [x \leftarrow ?] P)$ .

This obvious fact, which on the one hand permits elimination of quantifiers from random-DL completely, does not on the other hand imply that the presence of  $\exists x$  renders random assignments redundant;  $x \leftarrow ?$  can certainly appear, say, inside  $\forall y$  programs as we illustrate after the theorem below. In fact, here too we do not know whether or not any expressive power is gained.

*Open Problem:* Is DL  $\leq$  random-DL?

We do have the following result, which refers to DL with both array and random assignments.

*Theorem 2.3 (Meyer [44]):*

- (1) array-DL  $\leq$  random-array-DL,
- (2) random-DL  $\leq$  random-array-DL.

This result is proved by showing that there is no formula in DL with array-assignment or random-assignment (but not both) which is equivalent to the formula  $P: \langle \alpha \rangle \forall y \langle \beta \rangle true$ , where we define

$$\alpha: x \leftarrow z; (u \leftarrow ?; f(x) \leftarrow u; x \leftarrow f(x))^*, \text{ and}$$

$$\beta: x \leftarrow z; (x \leftarrow f(x))^*; (x = y)?; (x \leftarrow f(x))^*; (x = z)?$$

$P$  is a formula of this doubly augmented DL, which is true in a state  $J$  iff the domain of  $J$  is *finite*.  $\alpha$  makes possible assigning  $f(z)$ ,  $f(f(z))$  etc. to some random elements of the domain, and  $\beta$  makes sure that  $y$  is on the "f-cycle" starting from  $z$ . Finiteness, then, is definable in DL with both array- and random-assignment. It can be shown however, and this is the content of the remainder of the proof of Theorem 2.3, that finiteness is *not* definable in either array-DL or in random-DL.

### 2.3.3 Rich Test.

Rich-test-DL is the first-order version of  $PDL_{\infty}$  defined in Section 1.2. It allows tests in programs to involve other programs (which themselves might involve such tests etc.). Thus a program  $\alpha$  might pause, asking something like "can program  $\beta$  halt on input  $x$  if started right now?", and continue without side effects iff the answer was "yes".

The definition of rich-test-DL is identical to that of DL except that clause (2) in that definition is changed to read:

- (2) For any rich-test-DL-wff  $P$ ,  $P?$  is in RC.

So that, for example, a desired effect could be guaranteed "in advance" as in the program  $\alpha$ :  $(([\beta]P)?;\beta)^*$ , for which  $P \supset [\alpha]P$  is valid. Here  $\beta$  is not executed unless  $P$  is guaranteed to hold upon completion.

*Open Problem:* Is  $DL < \text{rich-test-DL}$ ?

### 2.3.4 Deterministic Dynamic Logic (DDL).

DDL is the deterministic version of DL; i.e. the only programs allowed inside boxes and diamonds are deterministic ones. We do this by defining the set of DDL-wffs to be simply the set of DL-wffs in which  $\cup$  and  $*$  appear only in constructs of the form  $(P?;\alpha \cup (\neg P)?;\beta)$  and  $((P?;\alpha)^*;\neg P?)$ , and we abbreviate these to (*if P then  $\alpha$  else  $\beta$* ) and (*while P do  $\alpha$* ) respectively. We call this restricted class of programs DRG, and clearly they correspond to the well known *while* programs. The semantics of DDL is the same as that of DL.

**Lemma 2.4:** For any universe  $U$ , state  $f \in U$  and program  $\alpha \in \text{DRC}$ , there is at most one state  $g \in U$  such that  $J\alpha g$ .

**Corollary 2.5:** The following are valid for every  $\alpha \in \text{DRC}$  and DL-wffs  $P$  and  $Q$ :

- (a)  $\langle \alpha \rangle P \equiv (\exists \alpha]P \wedge \langle \alpha \rangle \text{true})$ ,  
 (b)  $\langle \alpha \rangle (P \wedge Q) \equiv (\langle \alpha \rangle P \wedge \langle \alpha \rangle Q)$ .

*Proof:* We prove (a).  $J \vdash \langle \alpha \rangle P$  iff  $\exists g (J\alpha g \wedge g \vdash P)$  iff (by the lemma)  $\exists g (J\alpha g \wedge \forall g' (J\alpha g' \Rightarrow g' \vdash P))$  iff  $\exists g (J\alpha g \wedge g \vdash \text{true}) \wedge \forall g' (J\alpha g' \Rightarrow g' \vdash P)$  iff  $J \vdash (\langle \alpha \rangle \text{true} \wedge [\alpha]P)$ . ■

Here the question of whether nondeterminism supplies more expressive power is most interesting, and also unanswered as of now. Its clarification would hopefully supply insight into the proposal to employ nondeterminism in reasoning about programs.

**Open Problem:** Is  $\text{DDL} < \text{DL}$ ?

One can also define DPDL as PDL with similar restrictions upon the class of  $R$  of regular expressions over  $AP$ . There too we have:

**Open Problem:** Is  $\text{DPDL} < \text{PDL}$ ?

Note though, that the programs in DPDL can be nondeterministic by virtue of the interpretation assigning a non-functional relation to an atomic program. However, we can restrict the structures and ask the same question:

A binary relation  $r$  is said to be *functional* if for every  $a$  there is at most one  $b$  such that  $(a,b) \in r$ .

**Open Problem:** Is it the case that for every PDL-wff  $P$  there exists a DPDL-wff  $Q$  such that  $\models_S (P \equiv Q)$  for every structure  $S = (W, \sigma, m)$  in which  $m$  (restricted to  $AP$ ) is functional.

We now define the notion of total correctness which intuitively states that "the program will terminate satisfying the conditions":

**Definition:** A program  $\alpha$  in DRC is *totally correct* with respect to a universe  $U$  and DDL-wffs  $P$  and  $Q$ , if  $\vdash_U (P \Rightarrow \langle \alpha \rangle Q)$ .



Note that Corollary 2.5(a) substantiates the widely used fact that for deterministic programs, proving partial correctness and termination is the same as proving total correctness (see for example Manna [39]).

Thus, DL is a tool powerful enough to express the concept of total correctness for deterministic programs. However, in Chapter 5 we will see that this notion is much more subtle when nondeterministic programs are allowed.

Another interesting restriction on the programs in DL is the *guarded commands* language of Dijkstra [13]. We define this language in Section 5.5.

### 2.3.5 R.e. Dynamic Logic.

As it turns out (see for example Section 2.4), many interesting properties of dynamic logic are invariant under drastic changes to the complexity of the programs involved. To provide a definite class which can be thought of as a plausible "upper bound" on this complexity, we introduce r.e. programs.

A regular program of RC can be thought of as a regular set of strings over the basic alphabet of assignments and tests. It is easy to see that taking the meaning of these programs to be the union (over this set) of the binary relations obtained by composing the relations corresponding to the components of each string in order, is consistent with our definition of the meaning of the regular expressions over this alphabet. R.e.-DL is obtained in a similar way by adopting as programs *r.e. sets* of strings over the above alphabet and defining their meaning similarly. One particular way in which to *represent* these programs is to supply a description of the Turing machine which recognizes this r.e. set, along with the (finite) sets of assignments and tests involved. The semantics of r.e.-DL-wffs is then obtained analogously to that of DL.

Thus, these programs are so complex, that merely deciding at each point in the execution "what to do next" might take the full power of Turing machines. Nevertheless, it turns out that this complexity does not affect most of the results about the validity problem in DL.

## 2.4 The Validity Problem for DL.

In this section we state some results concerning the question of how hard it is to decide whether a given DL-wff is valid. Since a valid DL-wff is one which is true in every state of every universe, this is not, so to speak, a "domain dependent" question but rather a question involving the behavior of completely uninterpreted programs. Throughout this section, we will use the notation of Rogers [50] for indicating degrees of undecidability.

The first fact about DL is the well-known recursive-enumerability of the set of valid first-order formulae:

*Lemma 2.6:* The valid program-free DL-wffs form a  $\Sigma_1^0$ -complete set.

*Proof:* These are precisely the valid first-order wffs of ordinary logic. ■

*Lemma 2.7 (Pratt [52]):* The valid DL-wffs with no appearance of the  $*$  operator, form a  $\Sigma_1^0$ -complete set.

*Proof:* Trivial, using Lemmas 1.1 and 2.1 for getting rid of these loop-free programs. ■

*Theorem 2.8 (Meyer and Pratt [22]):* The valid DL-wffs of the form  $\langle \alpha \rangle P$ , where  $P$  is first-order and  $\alpha$  is any r.e. program form a  $\Sigma_1^0$ -complete set.

In other words attaching one diamond (even with the most complicated program in it) to a first-order formula, does not make the validity problem any more difficult. In particular, one can extend this result to formulae of the form  $P \rightarrow \langle \alpha \rangle Q$  for program-free  $P$  and  $Q$ , showing that deciding validity of total correctness assertions for deterministic uninterpreted programs is an r.e. problem.

*Theorem 2.9 (Meyer and Pratt [22]):* The valid DL-wffs of the form  $\langle \alpha \rangle P$ , where  $P$  is program free and the set of programs is taken to be as large as the set of r.e. programs or as small as the singleton  $\{ x+y; (x+f(x))^* \}$ , form a  $\Pi_2^0$ -complete set.

Thus, attaching one box to a first-order formula gives rise to a very hard validity problem (as hard, in fact, as the totality problem for Turing machines). (Similarly, one can extend this to the class of valid partial correctness assertions.) However, if the

formula  $P$  to which  $[\alpha]$  is attached (the output specification of the partial correctness assertion) is free of existential quantifiers, i.e. is a *universal* formula, the problem is easier:

*Theorem 2.10* (Meyer and Pratt [22]): The valid DL-wffs of the form  $[\alpha]P$ , where  $\alpha$  is as in Theorem 2.9 and  $P$  is a universal first-order formula, form a  $\Pi_1^0$ -complete set.

The hopes of keeping the validity problem for the whole of DL down to some place in the arithmetic hierarchy are shattered by the following theorem:

*Theorem 2.11* (Meyer, [22] and [44]): The valid DL-wffs of each of the following forms, form a  $\Pi_1^1$ -complete set, where the set of programs involved can, in each case, be taken to be as large as the set of r.e. programs or as small as the singleton  $\{x \leftarrow y; (x \leftarrow f(x))^*\}$ :

- |   |  |
|---|--|
| (a) $\exists x[\alpha]P$                      | $P$ a first-order formula,                 |
| (b) $\exists x\exists y[\alpha]P$             | $P$ a quantifier-free first-order formula, |
| (c) $\langle\beta_1; \beta_2\rangle[\alpha]P$ | $P$ a quantifier-free first-order formula, |
| (d) $P$                                       | $P$ a DL-wff.                              |

Thus, the validity problem for DL is extremely hard, in fact as hard as deciding the validity of general universal second order formulae of the form  $\forall xP$ , where  $P$  is a first-order formula of arithmetic. It gets that way however, for quite simple formulae with only one "alternation" of programs (here we like to view  $\exists x$  as  $\langle x \leftarrow ? \rangle$ ). The upper bound of  $\Pi_1^1$  can be shown to hold for all the variations we have considered, in particular, the set of valid formulae of rich-test-random-array-DL also form a  $\Pi_1^1$ -complete set.

These results then, eliminate any possibility of obtaining (absolutely) complete axiomatizations of any interesting portions of DL. In the next chapter we will see however, that the situation is not so grim.

We remark here that Meyer [44] has also been able to show that the set of valid formulae of Salwicki's [59] *algorithmic logic* is also  $\Pi_1^1$ -complete. This is contrary to erroneous results in Kreczmar [32] and [33].

### 3. Arithmetical Axiomatization.

In this chapter we introduce the approach of supplying a syntactic characterization of the  $\mathcal{U}$ -valid DL-wffs for specific universes  $\mathcal{U}$ , namely those which "contain" the universe of arithmetic. This characterization will be in the form of a sound axiom system  $\mathcal{P}$  for DL which makes explicit use of variables that range over the natural numbers. For any such "arithmetical" universe  $\mathcal{A}$  we take all  $\mathcal{A}$ -valid first order formulae as axioms, and show that then  $\mathcal{P}$  is  $\mathcal{A}$ -complete, i.e. that a proof in  $\mathcal{P}$  does indeed exist for any  $\mathcal{A}$ -valid DL-wff. This property we term *arithmetical completeness*.

As will become evident in the sequel, the natural numbers are used in first order formulae to "count" the number of times  $\alpha$  is executed in  $\alpha^*$ . We do not use the extra power in which we indulge in order to introduce "arithmetical assignments" into the programs, i.e. assignments to variables which range over the natural numbers, as is done e.g. by Owicki [47] for reasoning about parallel programs. In fact, one's programs might not involve integers at all and still, since the particular universe which is being used can be extended to an arithmetical one, reasoning about these programs, without modification, can be carried out via an arithmetically complete system such as  $\mathcal{P}$ .

Anticipating our need for proving arithmetical completeness again in Chapters 4, 6 and 7, we state and prove a rather general theorem in Section 3.1, one which is a generalization of the induction step which we need for our completeness theorem in Section 3.2. This step is common to the completeness theorems for all the logics we consider in this thesis, and in fact we envision it as taking care of a major part of the proofs of similar completeness results in the future. Section 3.1 also includes the definition of an arithmetical universe, noting that any universe can be extended to an arithmetical one. It is then proved that for any arithmetical universe there exists, for every DL-wff, a first order formula equivalent to it over that universe. Section 3.2 contains our axiom system  $\mathcal{P}$  for DL and proofs of its arithmetical soundness and completeness. Section 3.3 contains the restriction  $\mathcal{D}\mathcal{P}$  of  $\mathcal{P}$  for dealing with DDL (see Section 2.3.4). In Section 3.4 we remark on the relationships holding between our approach, Cook's [12] *relative completeness*, and Mirkowska's [41] infinitary axiomatizations.

### 3.1 The Theorem of Completeness and Arithmetical Universes.

In this section we prove a general theorem which will be applied five times in the thesis for obtaining completeness results for arithmetical axiomatizations of various logics of programs. It will allow us to deduce, for example, the arithmetical completeness of an axiom system for DL given that that system is complete for proving basic formulae involving at most one program. The theorem, however, will be stated in very general terms.

Denote the set of first-order formulae by  $L$ . Assume we are given a universe  $U$ , a set  $K$ , and a functional

$$M: K \times 2^U \rightarrow 2^U.$$

The  $M$ -extension of  $L$ ,  $L(M)$ , is defined to be the following language which is  $L$  augmented with one formation-rule:

- (1) Any atomic formula is in  $L(M)$ ,
- (2) For any  $k \in K$ , variable  $x$  and  $L(M)$ -wffs  $P$  and  $Q$ ,  
 $\neg P$ ,  $(P \vee Q)$ ,  $\exists x P$  and  $(M_k)P$  are  $L(M)$ -wffs.

The semantics of  $L(M)$  are defined such that  $\mathcal{J} \models (M_k)P$  holds whenever  $\mathcal{J} \in \mathcal{M}(k, \{\mathcal{J} \mid \mathcal{J} \models P\})$ ; all the other clauses receive their standard meanings.

Some intuition might be gained at this point by noticing that if  $K$  is taken to be the class of programs  $RG$  and  $(M_\alpha)P$  is interpreted as  $\langle \alpha \rangle P$ , then  $L(M)$  is in fact regular first order dynamic logic, i.e. DL.

We now define some important concepts to be used in the sequel:

We say that  $L$  is  $U$ -expressive for  $L(M)$  if for every  $L(M)$ -wff  $P$  there exists an  $L$ -wff  $Q$  such that  $\models_U P \equiv Q$ .

An axiom system  $\mathcal{P}(M)$  for  $L(M)$  is any set of axioms (or axiom schemas) and inference rules over  $L(M)$ . Provability of an  $L(M)$ -wff  $P$  in  $\mathcal{P}(M)$  is defined in the standard way and is denoted by  $\vdash_{\mathcal{P}(M)} P$ .  $\mathcal{P}(M)$  is said to be  $U$ -sound if all the axioms are  $U$ -valid and all

the rules of inference preserve U-validity. Note then, that if  $P(M)$  is U-sound, then whenever  $\vdash_{P(M)} R$  holds,  $\vdash_U R$  does too.

$P(M)$  is said to be *propositionally complete* if all instances of tautologies of propositional calculus are theorems of  $P(M)$  and *modus ponens* is in the set of inference rules. It is said to be *U-complete* if for every  $L(M)$ -wff  $R$ , if  $\vdash_U R$  holds then we have  $\vdash_{P(M)} R$ .

**Theorem 3.1 (Theorem of Completeness):** For any universe  $U$  and  $M$ -extension  $L(M)$  of  $L$ , a U-sound axiom system  $P(M)$  for  $L(M)$  is U-complete whenever

- (1)  $P(M)$  is propositionally complete,
- (2)  $L$  is U-expressive for  $L(M)$ ,
- (3) For any  $k \in K$  and  $L(M)$ -wffs  $R$  and  $Q$ ,  
if  $\vdash_{P(M)} (R \supset Q)$  then  $\vdash_{P(M)} ((M_k)R \supset (M_k)Q)$ , and
- (4) For any  $k \in K$  and  $L$ -wffs  $R$  and  $Q$ ,  
if  $\vdash_U R$  then  $\vdash_{P(M)} R$ ,  
if  $\vdash_U (R \supset (M_k)Q)$  then  $\vdash_{P(M)} (R \supset (M_k)Q)$ , and  
if  $\vdash_U (R \supset \neg(M_k)Q)$  then  $\vdash_{P(M)} (R \supset \neg(M_k)Q)$ .

*Proof:* We have to prove that if  $P$  is an  $L(M)$ -wff such that  $\vdash_U P$ , then  $\vdash_{P(M)} P$ .

By the propositional completeness of  $P(M)$  we can assume that  $P$  is given in conjunctive normal form, and we proceed by induction on the sum of the number of appearances of  $M$  and the number of quantifiers in  $P$ . Assume the theorem holds for any formula with  $n-1$  or less appearances of  $M$  and quantifiers. If  $P$  is of the form  $P_1 \wedge P_2$  then we have  $\vdash_U P_1$  and  $\vdash_U P_2$ , both of which have to be proved in  $P(M)$ , so that we can restrict our attention to a single disjunction. Without loss of generality we can, therefore, assume that  $P$  is of one of the forms:

$$P_1 \vee (M_k)P_2, \quad P_1 \vee \neg(M_k)P_2, \quad P_1 \vee \exists x P_2 \quad \text{or} \quad P_1 \vee \exists x \neg P_2,$$

where  $k \in K$  and  $P_1$  and  $P_2$  each have  $n-1$  or less appearances of  $M$  and quantifiers. Let us use  $\rho$  to denote  $(M_k)$ ,  $\neg(M_k)$ ,  $\exists x$  or  $\exists x \neg$  according to which is the case.

$L$  is expressive for  $L(M)$ , and so for any  $L(M)$ -wff  $Q$  there is some  $L$ -wff  $Q_L$  which is equivalent to  $Q$ . We have then  $\vdash_U (\neg P1_L \supset \rho P2_L)$ . Now, using assumption (4) (since  $P1_L$  and  $P2_L$  are  $L$ -wffs) we also have

$$(*) \quad \vdash_{P(M)} (\neg P1_L \supset \rho P2_L).$$

Now surely, by the definition of  $P1_L$  and  $P2_L$ , we have  $\vdash_U (\neg P1 \supset \neg P1_L)$  and  $\vdash_U (P2_L \supset P2)$ . Both these last formulae have less than  $n$  appearances of  $M$  and quantifiers, and hence by the inductive hypothesis

$$(**) \quad \vdash_{P(M)} (\neg P1 \supset \neg P1_L) \quad \text{and} \\ \vdash_{P(M)} (P2_L \supset P2).$$

By assumption (3) or the first clause in (4) (depending on whether  $\rho$  is an appearance of  $M$  or a quantifier) together with the propositional completeness, we obtain from the latter

$$(***) \quad \vdash_{P(M)} (\rho P2_L \supset \rho P2).$$

From (\*), (\*\*), and (\*\*\*) we get, using propositional reasoning,  $\vdash_{P(M)} (\neg P1 \supset \rho P2)$ , or  $\vdash_{P(M)} (P1 \vee \rho P2)$ . ■

Our goal in the next section is to apply this theorem to DL viewed as an  $M$ -extension of  $L$  as indicated above. In order to do this we now define a set of universes, the arithmetical universes, each of which satisfies requirement (2) of the Theorem. This fact is proved below in Theorem 3.2.

An *arithmetical universe*  $A$  is a universe in which the domain includes the set of natural numbers, the binary function symbols  $+$  and  $\cdot$  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 " $nat_j(d)$  is true iff  $d$  is a natural number", that is, for every state  $j$   $\{d \in D_j \mid nat_j(d)\}$  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  $j$  when it is not the case that  $nat_j(x_j)$  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

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

The intuition is that  $R(x, i, y)$  holds iff " $x$  is the  $i$ 'th component of  $y$ ", so that any finite sequence  $x_1 \dots x_n$  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  $+$ ,  $\cdot$ ,  $0$ ,  $=$  and  $\text{nat}$  (which in this case is identically true), are the only function and predicate symbols. Code's  $\beta$ -function (see e.g. Section 6.4 in Shoenfield [62]) serves as the finite sequence encoding function.

It is important to note that any universe  $U$  can be extended to an arithmetical universe  $A_U$  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.

Take  $A$  to be any arithmetical universe and  $K$  to be the set  $RC$  of regular programs over assignments and tests. Take  $M$  to be the "diamond" operator, or more precisely define  $M(\alpha, \{J \mid J \models P\}) = \{J \mid \exists J' (\exists \alpha J' \wedge J' \models P)\}$ , so that  $J \models (M_\alpha)P$  iff  $J \models \langle \alpha \rangle P$ . Certainly then,  $L(M)$  is simply DL.

We remark here that in fact we will be using Theorem 3.1 in the sequel only with functionals  $M$  (such as  $\langle \alpha \rangle$ ) for which we are interested in at most one of the states satisfying  $P$ . Consequently, we could have defined  $M: K \times U \rightarrow 2^U$ , and then  $J \models (M_\alpha)P$  iff  $\exists J' (J' \models P \wedge M(\alpha, J'))$ . However, we have stated and proved the theorem in this general form to facilitate possible future applications of it.



**Theorem 3.2:** L is A-expressive for DL.

*Proof:* We have to show that for every DL-wff P there exists an L-wff  $P_L$  such that  $\models_A (P \equiv P_L)$ . We proceed by induction on P. The cases where P is an atomic formula, or of one of the forms  $\neg Q$ ,  $Q \vee R$  or  $\exists x Q$  are straightforward. Assume P is of the form  $\langle \alpha \rangle Q$  for  $\alpha \in RC$  and assume  $Q_L$  is the L-wff which is A-equivalent to Q. Denote  $var(\alpha)$  by Z, and by Z' denote a vector of the same length as Z whose components are distinct variables not in  $var(\alpha)$ . By convention we can denote by  $x'$  the element of Z' corresponding to an element x of Z. We show, by induction on the structure of  $\alpha$ , that there exists an L-wff  $F_\alpha(Z, Z')$  such that for any DL-wff Q we have

$$(*) \quad \models_A (\langle \alpha \rangle Q \equiv \exists Z' (F_\alpha \wedge (Q_L)_{Z'}^{Z'})),$$

where  $(Q_L)_{Z'}^{Z'}$  is the obvious generalization of  $(Q_L)_x^x$  to vectors of variables. Thus in a sense, we find a formula  $F_\alpha$  which is true of Z and Z' iff  $\alpha$  can "change" the value of Z to that of Z'.

For an assignment take  $F_{x \leftarrow e}$  to be  $x' = e$ . Surely  $\langle x \leftarrow e \rangle Q$  is A-equivalent to  $\langle x \leftarrow e \rangle Q_L$  which is A-equivalent to  $(Q_L)_x^e$ , or in fact to  $\exists x' (x' = e \wedge (Q_L)_x^{x'})$ .

For the case where  $\alpha$  is of the form  $\beta \cup \beta'$ , take  $F_{(\beta \cup \beta')}$  to be  $(F_\beta \vee F_{\beta'})$ . Similarly, when  $\alpha$  is  $\beta ; \beta'$ ,  $F_{(\beta ; \beta')}$  is taken to be  $\exists Z'' ((F_\beta)_{Z''}^{Z''} \wedge (F_{\beta'})_{Z''}^{Z''})$ . Here Z'' is a "fresh" vector like Z'. It is quite straightforward to verify that (\*) holds for both these cases.

Assume  $\alpha$  to be of the form  $\beta^*$ . By standard techniques, using the encoding of finite sequences into single elements of the domain, we can construct an "iteration" formula  $ITR_\beta$  with a free variable, such that we have  $ITR_\beta(0) \equiv (Z=Z')$ , where  $Z=Z'$  abbreviates the conjunction of the equality of the corresponding components of Z and Z',  $ITR_\beta(1) \equiv F_\beta$ , and for any natural number  $n > 1$  we have (slightly abusing strict notation):

$$ITR_\beta(n) \equiv (\exists Z_1) \dots (\exists Z_{n-1}) ((F_\beta)_{Z_1}^{Z_1} \wedge ((F_\beta)_{Z_1 Z_2}^{Z_1 Z_2} \wedge ((F_\beta)_{Z_1 Z_2 Z_3}^{Z_1 Z_2 Z_3} \wedge \dots \wedge (F_\beta)_{Z_1}^{Z_{n-1}})).$$

It is then easy to see that for any n,  $\langle \alpha^n \rangle Q$  is A-equivalent to  $(\exists Z' (ITR_\beta(n) \wedge (Q_L)_{Z'}^{Z'}))$ , and hence that  $F_{\beta^*}$  can be taken to be  $(\exists n) (nat(n) \wedge ITR_\beta(n))$ , and that then (\*) will hold. ■

Thus by inspecting the assumptions of Theorem 3.1 in this context we arrive at the conclusion that if we can find an  $A$ -sound axiom system  $P$  for DL, such that

- (a)  $P$  is propositionally complete,
- (b)  $P$  includes all  $A$ -valid first-order wffs as axioms,
- (c)  $P$  includes the inference rule

$$\frac{R \supset Q}{\langle \alpha \rangle R \supset \langle \alpha \rangle Q},$$

- and (d) we can prove completeness of  $P$  for formulae of the simple forms  $R \supset \langle \alpha \rangle Q$  and  $R \supset \langle \alpha \rangle Q$  with first order  $R$  and  $Q$ ,

then indeed by Theorem 3.1 we have an  $A$ -sound and  $A$ -complete axiom system for DL. An axiom system which, for every arithmetical universe  $A$  when augmented with all  $A$ -valid L-wffs as axioms is  $A$ -sound and  $A$ -complete, is called *arithmetically complete*. In the next section we set ourselves out to find such an arithmetically complete axiom system for DL.

## 3.2 Axiomatization of DL.

In this section we provide an arithmetically complete axiom system  $P$  for DL. In the sequel  $A$  stands for any arithmetical universe, and  $L$  for the set of first-order formulae. When talking about arithmetical universes we will often want to use  $n, m, \dots$  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 " $\forall n(n) \supset$ ". Thus, for example,  $\exists P(P(n) \supset Q)$  stands for  $\exists P(\forall n(n) \supset (P(n) \supset Q))$ , assuming that in state  $S$ ,  $(P(n) \supset Q)$  is true if  $n$  happens to be a natural number. Furthermore, by convention,  $\forall n P(n)$  stands for  $\forall n(\forall n(n) \supset P(n))$ , and hence  $\exists n P(n)$  abbreviates  $\exists n(\forall n(n) \wedge P(n))$ .

Consider the following axiom system  $P$  for DL:

*Axioms:*

- (A) All tautologies of propositional calculus.
- (B) All  $A$ -valid L-wffs.
- (C)  $\langle x \rightarrow e \rangle P = P_x^e$ , for an L-wff  $P$ .

- (D)  $[Q?]P \equiv (Q \supset P)$ .  
 (E)  $[\alpha; \beta]P \equiv [\alpha][\beta]P$ .  
 (F)  $[\alpha \cup \beta]P \equiv ([\alpha]P \wedge [\beta]P)$ .

*Inference rules:*

$$(G) \frac{P, P \supset Q}{Q}$$

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

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

$$(J) \frac{P(n+1) \supset \langle \alpha \rangle P(n)}{P(n) \supset \langle \alpha^* \rangle P(0)} \quad \text{for an L-wff } P \text{ with free } n, \text{ s.t. } n \notin \text{var}(\alpha).$$

Rules (I) and (J) are called the rules of *invariance* and *convergence* respectively.

A DL-wff  $P$  is said to be *provable* in  $P$ , written  $\vdash_P P$ , if there exists a finite sequence  $S$  of DL-wffs the last one being  $P$  and such that each formula in  $S$  is an axiom (or instance of an axiom scheme) or is obtained from previous formulae of  $S$  by one of the rules of inference.

We first establish the soundness of the inference rules which appear in  $P$ :

*Lemma 3.3:* For any universe  $U$ , DL-wffs  $R$  and  $Q$ , and  $\alpha \in RC$ ,  
 if  $\vDash_U R \supset Q$  then  $\vDash_U ([\alpha]R \supset [\alpha]Q)$ .

*Proof:* Assume  $\vDash_U R \supset Q$ , and  $\mathcal{J} \vDash [\alpha]R$  for some  $\mathcal{J} \in U$ . Thus for every  $\mathcal{J} \in U$  such that  $\mathcal{J} \vDash \alpha$  we have  $\mathcal{J} \vDash R$ . Surely then, from  $\mathcal{J} \vDash R \supset Q$  we have  $\mathcal{J} \vDash Q$ . Thus,  $\mathcal{J} \vDash [\alpha]Q$ . ■

**Lemma 3.4:** For any universe  $U$ , DL-wff  $P$  and  $\alpha \in RC$ , if  $\vdash_U (P \supset [\alpha]P)$   
then  $\vdash_U (P \supset [\alpha^*]P)$ .

*Proof:* Assume  $\vdash_U (P \supset [\alpha]P)$  and  $J \vdash P$  for some  $J \in U$ . We have to show  $J \vdash [\alpha^n]P$  for all  $n$ . We proceed by induction on  $n$ . For  $n=0$   $J \vdash [\alpha^0]P$  if  $J \vdash \langle true \rangle P$  if  $J \vdash (true \supset P)$  if  $J \vdash P$  which is assumed. Assume  $J \vdash [\alpha^n]P$ . By  $\vdash_U (P \supset [\alpha]P)$  we can obtain  $\vdash_U ([\alpha^n]P \supset [\alpha^n][\alpha]P)$ , and then conclude  $J \vdash [\alpha^n][\alpha]P$  or  $J \vdash [\alpha^{n+1}]P$ . ■

**Lemma 3.5:** For any L-wff  $P(n)$  and  $\alpha \in RC$ , where  $n \neq var(\alpha)$ ,  
if  $\vdash_A (P(n+1) \supset \langle \alpha \rangle P(n))$  then  $\vdash_A (P(n) \supset \langle \alpha^* \rangle P(0))$ .

*Proof:* Assume  $\vdash_A (P(n+1) \supset \langle \alpha \rangle P(n))$  and  $J \vdash P(n)$ . We show  $J \vdash \langle \alpha^* \rangle P(0)$  or  $J \vdash \exists n \langle \alpha^n \rangle P(0)$  by induction on  $n_j$ . For  $n_j=0$  we have  $J \vdash (true \wedge P(0))$  or  $J \vdash \langle true \rangle P(0)$  which is  $J \vdash \langle \alpha^0 \rangle P(0)$ . Assume that  $J \vdash \langle \alpha^m \rangle P(0)$  holds whenever  $J \vdash P(m)$  and  $m_j = n_j - 1$ . By  $\vdash_A (P(n+1) \supset \langle \alpha \rangle P(n))$  we conclude  $\exists j (J \alpha j \wedge J \vdash P(n))$  and  $n_j = n_j - 1$ . But then  $J \vdash \langle \alpha^m \rangle P(0)$ , from which we have  $J \vdash \langle \alpha \rangle \langle \alpha^m \rangle P(0)$  or  $J \vdash \langle \alpha^* \rangle P(0)$ . ■

We remark here that the rule of invariance (I) can be replaced by the induction axiom scheme

$$[\alpha^*](P \supset [\alpha]P) \supset (P \supset [\alpha^*]P),$$

which is derivable from  $P$ , and from which, in  $P$ , rule (I) can be derived.

**Theorem 3.6 (A-soundness of  $P$ ):** For any DL-wff  $P$ , if  $\vdash_P P$  then  $\vdash_A P$ .

*Proof:* Follows from Lemmas 1.1, 1.7, 2.1, 3.3, 3.4 and 3.5. ■

We now apply the general Theorem of Completeness of the previous section to obtain an arithmetical completeness result for  $P$ . However, in order to apply that theorem we have to prove that  $P$  is A-complete for formulae of the forms  $R \supset [\alpha]Q$  and  $R \supset \langle \alpha \rangle Q$  with program-free  $R$  and  $Q$ . These two results, Box-completeness (Theorem 3.9) and Diamond-completeness (Theorem 3.11) are obtained analogously. They are both proved by induction on the structure of  $\alpha$ . The difficulty is when  $\alpha$  is of the form  $\beta^*$ , in which case we show that when, say,  $R \supset [\beta^*]Q$  is A-valid, then there is a way of proving that fact in  $P$ . This is done by exhibiting derived rules (I') and (J') below to cover these cases, and proving that they can be applied.

**Lemma 3.7:** The following are derived rules of  $P$ :

$$(H') \quad \frac{P \supset Q}{\langle \alpha \rangle P \supset \langle \alpha \rangle Q}$$

$$(I') \quad \frac{R \supset P, P \supset [\alpha]P, P \supset Q}{R \supset [\alpha^*]Q}$$

$$(J') \quad \frac{R \supset \exists n P(n), P(n+1) \supset \langle \alpha \rangle P(n), P(0) \supset Q}{R \supset \langle \alpha^* \rangle Q} \quad \begin{array}{l} P \text{ and } n \text{ as} \\ \text{in rule (J).} \end{array}$$

*Proof:* (H'): From  $\vdash_P (P \supset Q)$  we obtain, using (A) and (G),  $\vdash_P (-Q \supset -P)$ . Apply (H) to get  $\vdash_P ([\alpha]-Q \supset [\alpha]-P)$ , then (A) and (G) to obtain  $\vdash_P (\langle \alpha \rangle P \supset \langle \alpha \rangle Q)$ .

(I'): From  $\vdash_P (P \supset [\alpha]P)$  we have by (I)  $\vdash_P (P \supset [\alpha^*]P)$ , and then using  $\vdash_P (R \supset P)$  and (A) and (G), we obtain  $\vdash_P (R \supset [\alpha^*]P)$ . From  $\vdash_P P \supset Q$  and (H) we have  $\vdash_P ([\alpha^*]P \supset [\alpha^*]Q)$  and thus again with (A) and (G),  $\vdash_P (R \supset [\alpha^*]Q)$ .

(J'): Like (I') but using the fact that from  $\vdash_P (R \supset \exists n P(n))$  and  $\vdash_P (P(n) \supset \langle \alpha^* \rangle Q)$  we can deduce  $\vdash_P (R \supset \langle \alpha^* \rangle Q)$  using (B), (A) and (G). ■

An L-wff  $P$  which A-validates the premises of (I') is called an *invariant* of  $\alpha$  with respect to  $R$  and  $Q$ . The concept of invariance has been studied quite extensively in the literature on program verification, see for example [29]. An L-wff  $P(n)$  which A-validates the premises of (J') we term a *convergent* of  $\alpha$  with respect to  $R$  and  $Q$ . This concept does not seem to have received adequate treatment.

We now show that it is always possible to find an invariant of  $\alpha$  wrt  $R$  and  $Q$ , under the assumption that the conclusion of rule (I') is A-valid.

**Lemma 3.8 (Invariance Lemma):** For every  $\alpha \in RC$  and DL-wffs  $R$  and  $Q$ , if  $\vdash_A (R \supset [\alpha^*]Q)$  then there exists an L-wff  $P$  such that  $\vdash_A (R \supset P)$ ,  $\vdash_A (P \supset [\alpha]P)$  and  $\vdash_A (P \supset Q)$ .

*Proof:* By Theorem 3.2 there is an L-wff  $P$  which is A-equivalent to  $[\alpha^*]Q$  (i.e.  $\vdash_A (P \equiv [\alpha^*]Q)$ ). Certainly by  $\vdash_A (R \supset [\alpha^*]Q)$  we have  $\vdash_A (R \supset P)$ . Similarly, it is easy to see that  $\vdash_A (P \supset Q)$  and  $\vdash_A (P \supset [\alpha]P)$ . ■

**Theorem 3.9 (Box-completeness Theorem):** For every  $\alpha \in \text{RC}$  and L-wffs  $R$  and  $Q$ , if  $\vdash_A (R \supset [\alpha]Q)$  then  $\vdash_P (R \supset [\alpha]Q)$ .

*Proof:* We proceed by induction on the structure of  $\alpha$ . Assume the assertion of the theorem to hold for any  $\beta$  which is "smaller" than  $\alpha$  in the obvious inductive sense, and assume  $\vdash_A (R \supset [\alpha]Q)$ .

For  $\alpha$  an assignment or a test, (G) and (D) reduce the problem to that of "proving" an A-valid L-wff, which is simply an axiom.

If  $\alpha$  is  $\beta\cup\beta'$ , then proofs in  $P$  of  $(R \supset [\beta]Q)$  and  $(R \supset [\beta']Q)$  can be combined by (F) to a proof of  $R \supset [\beta\cup\beta']Q$ . Each of these being A-valid, we use the inductive hypothesis for both.

If  $\alpha$  is  $\beta;\beta'$  then we prove  $R \supset [\beta;\beta']Q$  in  $P$  in the following way and then use (E) to obtain the desired  $\vdash_P (R \supset [\beta;\beta']Q)$ : Certainly we have  $\vdash_A (R \supset [\beta;\beta']Q)$  and hence  $\vdash_A (R \supset [\beta]P)$ , where  $P$  is an L-wff which is equivalent to  $[\beta']Q$  (and exists by Theorem 3.2). However,  $R \supset [\beta]P$  being A-valid, we apply the inductive hypothesis to obtain  $\vdash_P (R \supset [\beta]P)$ . Similarly we can show  $\vdash_P (P \supset [\beta']Q)$ , and then  $\vdash_P (R \supset [\beta]P \supset [\beta;\beta']Q)$ , from which, using (A) and (C), we get  $\vdash_P (R \supset [\beta;\beta']Q)$ .

For the case when  $\alpha$  is  $\beta^*$ , we simply use Lemma 3.8 which guarantees the existence of an L-wff  $P$  which renders the premises of the derived rule (I') A-valid. By the inductive hypothesis these can be proved in  $P$ , and then one application of (I') yields the final result. ■

Similarly, under the assumption that the conclusion of (I') is A-valid, we can always find a convergent of  $\alpha$  wrt  $R$  and  $Q$ :

**Lemma 3.10 (Convergence Lemma):** For every  $\alpha \in \text{RC}$  and DL-wffs  $R$  and  $Q$ , if  $\vdash_A (R \supset \langle \alpha^* \rangle Q)$  then there exists an L-wff  $P(n)$  with  $n \notin \text{var}(\alpha)$ , such that  $\vdash_A (R \supset \exists n P(n))$ ,  $\vdash_A (P(n+1) \supset \langle \alpha \rangle P(n))$ , and  $\vdash_A (P(0) \supset Q)$ .

*Proof:* By the proof of Theorem 3.2 one can construct an L-wff  $P(n)$  such that for every state  $J \in A$  and natural number  $i$ , if  $n_j = i$  then  $\langle \alpha^i \rangle Q$  is equivalent in  $J$  to  $P(n)$ . This we can write (slightly abusing notation) as  $\vdash_A (\forall n)(\text{nat}(n) \supset (\langle \alpha^n \rangle Q \equiv P(n)))$ . Certainly by  $\vdash_A (R \supset \langle \alpha^* \rangle Q)$  we deduce  $\vdash_A (R \supset \exists n P(n))$ . Similarly, it is easy to see that the other A-validities hold too. ■

**Theorem 3.11 (Diamond-completeness Theorem):** For every  $\alpha \in RC$  and L-wffs  $R$  and  $Q$ , if  $\vdash_A (R \supset \langle \alpha \rangle Q)$  then  $\vdash_P (R \supset \langle \alpha \rangle Q)$ .

*Proof:* The proof follows that of Theorem 3.9, using the derived duals of (C)-(F), and using Lemma 3.10 instead of 3.8. ■

We can now conclude that, for DL-wffs, A-validity and provability in  $P$  are equivalent concepts:

**Theorem 3.12 (Arithmetical Soundness and Completeness for DL):** For any DL-wff  $P$ ,

$$\vdash_A P \quad \text{iff} \quad \vdash_P P.$$

*Proof:* One direction is Theorem 3.6, and the other follows from Theorems 3.1, 3.2, 3.9 and 3.11, together with the fact that (A), (B), (C) and (H) are part of  $P$ . ■

Theorem 3.12 is significant in that it shows that a very simple and elegant axiom system is sufficient for carrying out the (A-validity-preserving) translation of DL-wffs to formulae of arithmetic, in a structured manner. As we point out in Section 3.4.1, viewing the process of proving properties of programs as supplying a proof of a formula in an axiom system which takes all the validities of the underlying first-order language as axioms, is due to Cook [12]. This observation then, gives rise to viewing such axiom systems as mechanisms for carrying out this translation.

Appendix B contains a proof in  $P$ , of the A-validity of a nontrivial DL-wff which asserts the total correctness of an iterative version of McCarthy's [40] 91-function program.

We remark that  $P$  is also an arithmetically-complete system for rich-test-DL (see Section 2.3.3). Also, random-DL (2.3.2) is completely axiomatized by adding the axiom  $[x \leftarrow ?]P \equiv \forall x P$  to  $P$ , under the condition that in a universe  $A$ , the only  $x$ 's we allow in random assignment statements of the form  $x \leftarrow ?$ , are

uninterpreted ones. Pratt [52] has spelled out the axiom to be added to  $P$ , in order to completely axiomatize array-DL (2.3.1).

We also note here that we have used a "weakest antecedent" approach in proving our completeness theorem. This can be seen in our taking  $P$  in the proof of Lemma 3.8 (resp.  $P(n)$  in the proof of Lemma 3.10), to be  $A$ -equivalent to  $\langle \alpha^n \rangle Q$  (resp.  $\langle \alpha^n \rangle Q$ ). A different proof of Lemma 3.8 (but not of 3.10) exists, employing the dual "strongest consequent" approach. This proof involves taking  $P$  to be  $A$ -equivalent to  $\langle (\alpha^-)^n \rangle R$ , where  $m(\alpha^-)$  is defined as  $\{(J, J) \mid (J, J) \in m(\alpha)\}$ . A clarification of this observation appears in a wider context in Section 6.3.

### 3.3 A Derived Axiomatization of DDL.

In this section we supply an arithmetically complete axiom system  $DP$  for DDL (see Section 2.3.4) and compare it to the systems of Hoare [27] and Wang [69].  $DP$  is basically a "special case" of  $P$  in the sense that its axioms and rules are identical to, or are straightforwardly derived from, those of  $P$ . Nevertheless, our point in carrying out the synthesis of  $DP$  from  $P$  is precisely to exhibit the way in which special-purpose systems such as Hoare's can be derived from a system such as  $P$ .

Consider the following axiom system  $DP$  for DDL:

*Axioms:*

(A), (B), (C) and (E) as in  $P$ ,

(D', F')  $\langle \text{if } S \text{ then } \alpha \text{ else } \beta \rangle Q = ((S \Rightarrow \langle \alpha \rangle Q) \wedge (\neg S \Rightarrow \langle \beta \rangle Q))$ .

*Inference Rules:*

(G) and (H) as in  $P$ ,

(I'')  $(P \wedge S) \Rightarrow \langle \alpha \rangle P$

---

$P \Rightarrow \langle \text{while } S \text{ do } \alpha \rangle (P \wedge \neg S)$



$$(J'') \quad \frac{P(n+1) \supset (S \wedge \langle \alpha \rangle P(n)), \quad P(0) \supset \neg S}{P(n) \supset \langle \text{while } S \text{ do } \alpha \rangle P(0)}$$

$P$  and  $n$  as in rule (J),

Provability in  $DP$  is defined as usual.

**Lemma 3.13:** For any  $\alpha$  and  $\beta$  in  $RC$ , DL-wff  $Q$  and test  $S?$ , the following are valid:

- (1)  $\text{If } S \text{ then } \alpha \text{ else } \beta \text{ ]} Q \equiv ((S \supset \langle \alpha \rangle Q) \wedge (\neg S \supset \langle \beta \rangle Q)),$
- (2)  $\text{[while } S \text{ do } \alpha \text{ ]} Q \equiv \text{[(S?; } \alpha \text{)*]}(S \vee Q).$

*Proof:* Trivial from the definitions of the deterministic constructs in Section 2.3.4 and Lemmas 1.1 and 1.7. ■

We now show the soundness of rules (I'') and (J''):

**Lemma 3.14:** For any universe  $U$ , DL-wff  $P$ ,  $\alpha \in RC$  and test  $S?$ , if  $\vDash_U((P \wedge S) \supset \langle \alpha \rangle P)$  then  $\vDash_U(P \supset \langle \text{while } S \text{ do } \alpha \rangle (P \wedge \neg S)).$

*Proof:* We have  $\vDash_U(P \supset (S \supset \langle \alpha \rangle P))$  or  $\vDash_U(P \supset (S?; \alpha)P)$ . By Lemma 3.4 we have  $\vDash_U(P \supset [(S?; \alpha)*]P)$  and hence also  $\vDash_U(P \supset [(S?; \alpha)*](\neg S \supset (P \wedge \neg S)))$  which is simply  $\vDash_U(P \supset [(S?; \alpha)*; \neg S?](P \wedge \neg S)).$  ■

**Lemma 3.15:** For any L-wff  $P(n)$ , test  $S?$  and  $\alpha \in RC$ , where  $n \notin \text{var}(S?; \alpha)$ , if  $\vDash_A(P(n+1) \supset (S \wedge \langle \alpha \rangle P(n)))$  and  $\vDash_A(P(0) \supset \neg S)$  then  $\vDash_A(P(n) \supset \langle \text{while } S \text{ do } \alpha \rangle P(0)).$

*Proof:* By assumption we have  $\vDash_A(P(n+1) \supset \langle S?; \alpha \rangle P(n))$ , and so by Lemma 3.5 also  $\vDash_A(P(n) \supset \langle (S?; \alpha)* \rangle P(0))$ . By the second assumption we deduce that in fact  $\vDash_A(P(n) \supset \langle (S?; \alpha)* \rangle (\neg S \wedge P(0)))$  or  $\vDash_A(P(n) \supset \langle (S?; \alpha)*; \neg S? \rangle P(0)).$  ■

**Theorem 3.16 (Arithmetical Soundness and Completeness for DDL):** For any DDL-wff  $P$ ,

$$\vDash_A P \text{ iff } \vDash_{DP} P.$$

*Proof:* Soundness follows from Theorem 3.6 and Lemmas 3.13(1), 3.14 and 3.15. Completeness follows precisely in the footsteps of the proof of Theorems 3.9, 3.11 and 3.12, using the following two derived rules of  $DP$ :

$$(I'') \quad R \supset P, (P \wedge S) \supset \langle a \rangle P, (P \wedge \neg S) \supset Q$$

---


$$R \supset \langle \text{while } S \text{ do } a \rangle Q$$

$$(J'') \quad R \supset \exists n P(n), P(n+1) \supset (S \wedge \langle a \rangle P(n)), P(0) \supset (Q \wedge \neg S)$$

---


$$R \supset \langle \text{while } S \text{ do } a \rangle Q \quad \blacksquare$$

We remark that (I'') is precisely Hoare's [27] inference rule for proving the *partial correctness* of *while* programs. He writes  $P \langle a \rangle Q$  for  $P \supset (P \supset \langle a \rangle Q)$ . Also, (J'') is precisely one of Wang's [69] inference rules (rule T1 of [69]) for proving the *total correctness* of *while* programs. In fact,  $DP^*$  without rules (H) and (J'') represents a simple rephrasing of Hoare's [27] original system. We note here that we have shown both these rules to be derived in an easy way from the more general  $P^*$  in which the rules for  $a^*$  are concise, intuitively appealing, and quite easy to comprehend.

We refer the interested reader to the survey [21] in which we present more observations concerning other axiom systems and proof methods for reasoning about regular deterministic programs, which appear in the literature.

### 3.4 Related Work.

The approach to axiomatization taken in this thesis is closely related to, and was inspired by, Cook's [12] notion of relative completeness. In Section 3.4.1 we take up the task of comparing the two approaches. Section 3.4.2 is devoted to the description of the approach adopted by Mirkowicz [41] in her work on the algorithmic logic of Salwicki [59]. She uses infinitary inference rules in an axiom system, to characterize the valid (as opposed to  $\mathcal{U}$ -valid) formulae of this logic.

#### 3.4.1 Relative vs. Arithmetical Completeness.

As we indicated in the previous section, Hoare [27] introduced an axiom system for the partial correctness of programs, one which is basically a subsystem of  $DP^*$ . For the sake of this discussion we can in fact think of the corresponding subsystem of  $P^*$  consisting of (A), (C)-(G) and rule (I) as Hoare's system and denote it by  $H$ . Cook [12]

investigated the question of completeness of Hoare's system and managed to formalize what seems to be the intuitive way in which people prove correctness (partial in this case) of programs in line with the method suggested by Floyd [17] and Naur [46]. Cook separated the reasoning about the program from the reasoning about the underlying language, making a distinction between proving, say,  $[x \leftarrow 1]x=1$  and proving  $(x > 0 \rightarrow x \geq 0)$ . The former still requires some program-oriented manipulation in order to turn it into a first-order formula, whereas the second does not. Thus, Cook's idea was to supply Hoare's system with a generous oracle which had the ability to answer questions concerning the truth of first order formulae. In this way he was able to shift concentration to Hoare's rules themselves which were to serve as a tool for performing a step-by-step transformation of partial correctness assertions (of the form  $P \rightarrow [A]Q$ ) into equivalent first-order formulae. The truth of the latter is then checked using the oracle.

We now formally define Cook's [12] notion of relative completeness using the terminology we have developed. Assume given a language  $L'$  which includes all first-order formulae as wffs; thus  $L$  is part of  $L'$ . Assume  $AX$  is a sound axiom system for  $L'$  and denote by  $AX_U$  the system  $AX \cup \{B \vdash P\}$  and  $\vdash_U P$ . In other words,  $AX_U$  is  $AX$  augmented with all the  $U$ -valid first-order formulae as further axioms.  $AX$  is said to be *complete for  $L'$  relative to  $L$*  if for every universe  $U$  such that  $L$  is  $U$ -expressive for  $L'$ ,  $AX_U$  is  $U$ -complete for  $L'$  (every  $U$ -valid  $L'$ -wff is provable in  $AX_U$ ).

*Theorem 3.17 (Cook [12]):*  $H$  is complete for  $\{R \rightarrow [A]Q \mid R \text{ and } Q \text{ are } L\text{-wffs}\}$  relative to  $L$ .

The proof is in fact identical to that of our Box-completeness Theorem (Thm. 3.9).

Now, if we restrict ourselves to languages  $L'$  such that for any arithmetical universe  $A$ ,  $L$  is  $A$ -expressive for  $L'$ , we note that arithmetical completeness is a special case of relative completeness; we do not require that  $AX_U$  be  $U$ -complete for *all* universes  $U$  which make  $L$   $U$ -expressive for  $L'$ , but only that that be the case for any *arithmetical* universe. Consequently then, in  $AX$  itself we can use symbols in ways which take their standard interpretation for granted. This is the flavor of the usage of  $n$ ,  $+$  and  $0$  in the Rule of Convergence (rule (J) of  $P$ ).

The flurry of "positive" research which followed Cook's observation, and which was aimed at providing similar results for various extensions and variations of the programming language (eg. [19], [24] and [47]) led inevitably to a counter-effort of "negative" research aimed at proving incompleteness results which indicate when Hoare-like

systems are doomed to be incomplete even in the relative sense of Cook. The first notable result in this direction is that of Wand [67], who shows essentially that it is not the case that  $L$  is  $U$ -expressive for every universe  $U$ . Thus Wand shows that there exist universes  $U$  such that  $AX_U$  is not  $U$ -complete for  $L_H$ . More recently, Lipton [35] claims to have proved the following very interesting characterization of these "good" universes:  $L$  is  $U$ -expressive for  $L_H$  iff  $U$  is an arithmetical universe or a universe with a finite domain (call the latter a finite universe). Thus according to this claim the only universes for which a Hoare-like system can be relatively complete are the arithmetical ones and the finite ones. So Cook's [12] requirement boils down to requiring that  $AX_U$  be  $U$ -complete for these two kinds of universes.

The finite universes, however, cause trouble: Clarke [10] has shown that introducing (into the programming language in which the programs of  $L_H$  are written) various programming concepts such as procedures as parameters or coroutines, in the presence of recursion and other reasonable mechanisms, prevents the possibility of obtaining relatively complete axiom systems. The argument in [10] is based on the fact that the first order language  $L$  is  $U$ -expressive for  $L_H$  for any finite universe  $U$ . The incompleteness results are then established by showing that these complex programming languages have an undecidable halting problem over finite domains, and hence the set of diverging programs is not r.e., a fact which would contradict the existence of any relatively complete Hoare-like axiom system for such a language (the existence of one implying that, in particular, the set of valid formulae of the form  $true \Rightarrow \{a\} false$  is r.e.). Hence, the essence of Clarke's results lies in the fact that Cook's condition of expressiveness of  $L$  is satisfied by universes with finite domains.

The research of Lipton and Snyder [36] and Lipton [35] culminates in a generalization and extension of Clarke's results, with a theorem (Theorem 1 in [36]) which seems to tie up as equivalent the two properties of a programming language: (1) having a decidable halting problem over finite universes, and (2) the set of formulae  $P \Rightarrow \{a\} Q$  over it being r.e. in the set of all  $U$ -valid  $L$ -wffs, for any  $U$  such that  $L$  is  $U$ -expressive for  $L_H$ .

We conclude that relaxing the requirement and requiring that  $AX_U$  be  $U$ -complete only for all arithmetical universes (i.e. playing our arithmetical-completeness game) seems a reasonable thing to do even for the restricted language of partial correctness,  $L_H$ .

In addition, it seems that in order for axiomatizations of much richer logics like, say, DL (and the logics appearing in the sequel, CFDL, ADL,  $DL^+$  and CFDL $^+$ ) to be *relatively complete* (i.e. that they work for finite universes too), the rules that involve arithmetic (i.e. rule (J)) would have to be modified to deal with the finite-domain case, and would probably result in a system which is far less natural and elegant.

We are of the opinion, therefore, that the finite domains crept in because (1) the concept treated most extensively by researchers in the area was partial correctness ( $[A]P$  essentially), and (2) a weaker kind of expressiveness is needed to ensure the existence of an elegant relatively complete axiomatization of this particular concept on its own.

Thus we feel that it is natural and beneficial to allow the integers into ones reasoning language, in order to make possible the kind of "counting" we carry out in  $P$  (and later on in  $R$ ,  $P^+$  etc.).

Note that by adopting the "Hoare spirit" of structured, natural axiom systems, the remark in [67, pp. 90] "if the language is expressive it is trivial to write down a complete axiom system for partial correctness" becomes irrelevant. We are not interested in a one-rule system which has built into it essentially the full description of how to Godel-encode any wff and how to construct the equivalent formula of arithmetic. Rather, we want systems for composing our formulae step by step, using various kinds of assertions on the way. Of course, the proof that these systems are complete might involve relying on the expressive power of arithmetic, and hence might call upon the use of Godel encoding, in turn making "the formulae ... be less than perspicuous" [67] (as is the case with our completeness results which at various points require finding the arithmetical equivalent to formulae). Nevertheless, we believe that the construction of these systems contributes considerably to the understanding of the concepts involved and provides the framework in which the natural and intuitive proofs one might have for one's programs can be formulated.

### 3.4.2 Infinitary Axiomatization.

In 1970 Salwicki [59] introduced an *algorithmic logic* (AL) which is very close to DL in many respects, the main difference being that AL is designed to reason about *deterministic* regular programs only. Various directions of research were followed by the researchers at Warsaw initiated by Salwicki, and in particular *Mirkowska* [41] addressed

the problem of axiomatizing AL. (See [7] for a survey of their work and [21] for a comparison with DL.)

In this section we will not attempt to define AL, nor will we state any of the results relevant to it. We will, however, give a brief description of an infinitary axiom system IX for DL, derived from that of [41], and state a completeness theorem for it. This theorem is essentially due to Mirkowska, as one can carry over to DL the detailed proof (supplied in [42]) of the analogous theorem for AL which appears in [41].

The objective in constructing IX is entirely different from that of constructing P; the idea in IX is to provide a syntactical characterization of the *valid* DL-wffs, as opposed to the U-valid ones for specific universes U. Consequently, as we shall see, IX seems to be inadequate for proving properties of "interpreted" programs which operate over specific domains, and which use functions and predicates over these domains, having their standard interpretations in mind.

IX is an axiom system, which makes use of the following two tools for dealing with  $\alpha^*$ :

The axiom  $\langle \alpha^* \rangle P = (P \vee \langle \alpha \rangle \langle \alpha^* \rangle P)$ ,

and the rule

$$(oo) \quad \frac{\{ R \rightarrow [\alpha^i] Q \}_{i=0}^{\infty}}{R \rightarrow [\alpha^*] Q}$$

Besides these, IX includes the axioms (A), (D), (E) and (F), two rules for  $\forall x$ , the axiom  $[\alpha](P \supset Q) \supset ([\alpha]P \supset [\alpha]Q)$ , and a more complicated version of (C) catering for the case where P is a general DL-wff. Also, (C) is an inference rule of IX, as is the rule

$$\frac{P}{[\alpha]P} .$$

A *proof* of a DL-wff P in IX is a tree with root labeled by P, in which all paths are finite, and in which a node and its immediate ancestors are labeled in accordance with a rule of inference, the leafs being labeled with instances of axioms. Surely, by virtue of rule (oo), a proof-tree might be infinite; the crucial point, however, is that all paths are finite.

**Theorem 3.18 (Mirkowska [41]):** For every DL-wff  $P$ ,  $\models P$  iff  $\vdash_{IX} P$ .

Thus,  $IX$  characterizes the set of DL-wffs which are  $U$ -valid in every universe  $U$ .  $P$  on the other hand, is designed to characterize the sets of DL-wffs which are valid in arithmetical universes. Specifically, assume  $A$  is some arithmetical universe with uninterpreted function and predicate symbols. The set of  $A$ -valid DL-wffs and the set of  $A$ -valid first-order wffs are both  $\Pi_1^1$ -complete sets. Our axiom system  $P$  "gets its  $\Pi_1^1$  power" from axiom scheme (B), i.e. from taking the elements of the latter set as axioms. The rest of  $P$  then, can "afford" being finitary.  $IX$  also characterizes a  $\Pi_1^1$ -complete set, namely the set of valid DL-wffs (see Theorem 2.11), however it "gets its power" from the infinitary rule ( $\omega$ ) rather than from the set of axioms (which in the case of  $IX$  is r.e.). We can think of this situation as a trade-off between throwing the bulk of the  $\Pi_1^1$ -responsibility on the axioms or on the inference rules.

Another way of looking at the relationship is to note that since one can assert the existence of infinite trees, such as proofs in  $IX$ , using *finite* sentences of arithmetic, it is obvious that one can indeed give finitary inference rules to supplement a set of axioms which includes all valid sentences of arithmetic, and still be able to assert that a formula has an infinite proof in the  $IX$  sense.

Note for example, that the formula

$$(*) \quad \text{nat}(x) \supset \langle (x \leftarrow x-1)^* \rangle_{x=0}$$

is an  $A$ -valid wff, but not a valid one, and hence the reader should not be surprised that he cannot see how to prove it using the circular-looking axiom for  $\langle \alpha^* \rangle$  above. The *valid* wff which perhaps conveys the same idea as  $(*)$  is more complicated, and in it we have to replace  $\text{nat}(x)$  with a statement of the fact that  $x$  is accessible from  $z$  (standing for 0) via  $f$  (standing for successor), and that  $f$  acts on the set  $\{z, f(z), f(f(z)), \dots\}$  like successor does on the natural numbers:

$$(f(z) \neq z \wedge [y \leftarrow z; (y \leftarrow f(y))^*](g(f(y)) = y)) \supset [x \leftarrow z; (x \leftarrow f(x))^*](x \leftarrow g(x))^* \supset x = z.$$

This formula is valid, and provable in IX by virtue of each element of the set

$$\{ (f(z) \neq z \wedge [y \leftarrow z; (y \leftarrow f(y))^*](g(f(y)) = y)) \supset [x \leftarrow z; (x \leftarrow f(x))^i](x \leftarrow g(x))^* \supset x = z \}_{i=0}^{\infty}$$

being provable. This can be done for fixed  $i$  by applying the axiom above for  $\langle a^* \rangle$  exactly  $i$  times to  $\langle (x \leftarrow g(x))^* \rangle$ , thus "unraveling the loop" enough to obtain  $x = z$ . (In fact the proofs of each of these premises of rule (co) do not use (co) again.)



#### 4. Recursive Programs: Context-free Dynamic Logic (CFDL).

In this chapter we enrich the programming language we have been considering by replacing the  $*$  operator with a *recursion* operator on programs. Thus in a well defined sense we obtain *context-free* programs over assignments and tests as opposed to the *regular* ones we had previously.

The development of the material in this chapter is strongly affected by the analogy existing between, on the one hand, the concept of *iterating* as captured by the  $\alpha^*$  construct, and, on the other, that of *recursing* as captured by the simple recursive-program construct introduced below. The basic ideas present in the axiom systems appearing in [19] and [23] for proving the partial correctness of recursive programs are captured concisely by our box-rule for the recursive program construct, much as Hoare's [27] *while* rule is concisely captured by the rule of invariance of Section 3.3. Furthermore, we show that this rule is simply an instance of a principle of Park [51]. There is seemingly a drawback to our treatment in the fact that we do not provide tools for including any kinds of parameters in the programming language. The reason is in our wanting to achieve a clarification of the mechanisms for reasoning about *pure* recursion. Our experience in digesting the literature on this subject indicates that in most of the cases the presentation of the basic principles suffers from being obscured by rules for dealing with the parameters (i.e. rules of substitution, adaptation etc.). We consider one of the goals of this chapter the elimination of these rules and the exposition of the similarity between reasoning about iteration and recursion.

##### 4.1 Definitions.

The definition of CFDL is identical to that of DL, except that a different set of programs, namely CF, is employed instead of RG.

**Syntax:**

We assume given, besides the sets of symbols of Chapter 2, a set  $\Theta$  of *program variables*, elements of which we denote by  $X, X_1, X_2, \dots$ . The set of *program terms* is defined as follows:

- (1) Every assignment  $x \leftarrow e$ , test  $P?$  or program variable  $X \in \Theta$  is a term,
- (2) For all terms  $\tau_1, \dots, \tau_n$ , program variables  $X_1, \dots, X_n$  in  $\Theta$ , and for every  $i=1, \dots, n$ ,  $\tau_1; \tau_2$ ,  $\tau_1 \cup \tau_2$  and  $\mu_i X_1 \dots X_n (\tau_1, \dots, \tau_n)$  are terms.

The  $\mu_i X_1 \dots X_n (\tau_1, \dots, \tau_n)$  clause is intended, intuitively, to represent the program consisting of an execution of  $\tau_1$  where the appearances of the various  $X_j$  in the various  $\tau_k$  represent  $\tau_k$  calling  $\tau_j$ . Thus, we have  $n$  mutually recursive procedures. The bulk of this chapter, however, deals with the case  $n=1$  as described below.

An occurrence of  $X_j$  in a term  $\tau$  is said to be *bound* if it is in a subterm of the form  $\mu_i X_1 \dots X_n (\tau_1, \dots, \tau_n)$ , and *free* otherwise. A term with no free occurrences of any program variable is called *closed*. The set  $\mathbf{T}$  of *simple terms* is a subset of the set of terms, and is obtained by requiring that every subterm of the form  $\mu_i X_1 \dots X_n (\tau_1, \dots, \tau_n)$  is closed. The set  $\mathbf{T}_j$  of simple terms of width  $j$  is obtained from  $\mathbf{T}$  by restricting the value of  $n$  in any subterm of the form  $\mu_i X_1 \dots X_n (\tau_1, \dots, \tau_n)$  to be at most  $j$ .

The set  $\mathbf{CF}$  of *context-free* programs is taken to be simply the closed terms in  $\mathbf{T}_1$ . In Section 4.4 we sketch the extension of our results to the case where the set of programs is taken to be  $\mathbf{T} = \bigcup_{j=0}^{\infty} \mathbf{T}_j$ . At this point though, we can omit subscripts and, in the flavour of the semantics given below, can in fact adopt the convention of denoting  $\mu X \tau(X)$  by  $\tau^*(f)$ . Also, we have need only for one program variable  $X$  to serve as a "place holder". Thus,  $(y \rightarrow 1; N \cup y \in N)^*(f)$  is a legal program in  $\mathbf{CF}$ . *Context-free DL (CFDL)* is defined just as  $\mathbf{DL}$ , but using  $\mathbf{CF}$  instead of  $\mathbf{RC}$ .

**Semantics:**

All we really have to do here is define, for every  $\alpha \in \mathbf{CF}$ , the binary relation  $m(\alpha)$ , over the grand universe  $\mathbf{I}$ , which  $\alpha$  denotes. Inspection of the definition of  $\mathbf{CF}$

shows that in fact all we have to add to the definition of  $m$  in Chapter 2 is how to define  $m(\tau^*(f))$ .

For clarification we will sometimes write  $\tau(X)$  for a term  $\tau$  which has free occurrences of  $X$ , and no free occurrences of any other  $X' \in \Theta$ . Accordingly then, for such  $\tau$  we may take  $\tau(\alpha)$  to abbreviate  $\tau$  with all free occurrences of  $X$  replaced by the program  $\alpha$ .

Define  $\tau^0(\alpha) =_{df} \alpha$ , and  $\tau^{i+1}(\alpha) =_{df} \tau(\tau^i(\alpha))$ . Now define

$$m(\tau^*(f)) =_{df} \bigcup_{i=0}^{\infty} m(\tau^i(\text{false?})),$$

which to some extent explains our use of  $\tau^*(f)$  to denote  $\mu X \tau(X)$ .

*Example:* Consider the program

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

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

$$\begin{aligned} & ((z=0?; y \leftarrow 1) \cup (z \neq 0?; z \leftarrow z-1; \\ & \quad ((z=0?; y \leftarrow 1) \cup (z \neq 0?; z \leftarrow z-1; \\ & \quad \quad ((z=0?; y \leftarrow 1) \cup (z \neq 0?; z \leftarrow z-1; \\ & \quad \quad \quad \text{false?}; \\ & \quad \quad \quad z \leftarrow z+1; y \leftarrow y \cdot z)); \\ & \quad \quad z \leftarrow z+1; y \leftarrow y \cdot z)); \\ & \quad z \leftarrow z+1; y \leftarrow y \cdot z)). \end{aligned}$$

One can check that in any state  $J \in N$  for which  $x_J = 2$ , we have  $J \models \langle z \leftarrow x; \tau^3(\text{false?}) \rangle \text{true}$ ,

$J \models \langle z \leftarrow x; \tau^3(\text{false?}) \rangle y = 2$ , and for every  $n \neq 3$  we also have  $J \not\models \langle z \leftarrow x; \tau^n(\text{false?}) \rangle \text{false}$ .

Thus  $\alpha$ , given  $x=2$ , computes 2 in  $y$ . In general it can be seen that in the universe  $N$  of pure arithmetic, we have that  $m(\alpha)$  is the binary relation  $\{(J, J) \mid J = [(x_J)! / y] J\}$ , and thus  $\alpha$  is a program computing *factorial* over the natural numbers. ■

One can see then, that  $(J, J) \in m(\tau^*(f))$  iff there exists an integer  $n$  such that  $(J, J) \in m(\tau^n(\text{false?}))$ . 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"  $n$ . Thus, a successful

execution of the factorial program above, which is of the form  $z \leftarrow x; (\delta \cup \gamma; X; \beta)^*(f)$ , is any successful execution of  $z \leftarrow x; \gamma^i; \delta; \beta^l$  for some  $l$ .

(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  $\tau$ 's are all *continuous* over the domain of binary relations, and therefore defining the meaning of  $\mu_1 X_1 \dots X_n (\tau_1, \dots, \tau_n)$  to be the  $l$ 'th component of the *least* solution of the corresponding system of relational equations, in the sense of [4] and [26], is, by Kleene's [30] theorem, consistent with our definition of  $m(\tau^*(f))$ , or  $m(\mu X \tau(X))$ .)

In the sequel we will need some additional notation to aid in constructing our rules of inference and in conducting our meta-reasoning. Note that any program  $\alpha \in CF$  changes the values of at most the elements of  $var(\alpha)$ , all of which are variables. That is,  $\alpha$  cannot change the value of any second-order function symbol or of any predicate symbol. Consequently, we would like to make it possible to talk about binary relations, such as those represented by programs, in a first-order framework. We do this by defining an augmented programming language  $CF'$  in which there are programs corresponding to these relations.

Formally, the set  $CF'$  is defined as follows:

- For any L-wff  $P$  and vector of disjoint variables  $Z$ ,  $P^Z$  is in  $CF'$ .
- Any assignment  $x \leftarrow e$  or test  $P?$  is in  $CF'$ .
- Any closed term  $\tau^*(f) \in T_1$  is in  $CF'$ .
- For any  $\alpha, \beta \in CF'$ ,  $\alpha; \beta$  and  $\alpha \cup \beta$  are in  $CF'$ .

The meaning of  $P^Z$  is given by the following additional clause to the definition of  $m$ :

$$m(P^Z) = \{ (J, \mathcal{J}) \mid \mathcal{J} = [V/Z] \text{ for some vector } V \text{ of elements from } D_J, \text{ and } [Z_{\mathcal{J}}/Z] \models P \}.$$

Thus,  $P$  is thought of as having free variables  $Z$  and  $Z'$ , where  $Z'$  (in line with the remark in Section 2.2) is a vector of "primed versions" of the members of  $Z$ . Thus, 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  $J$   $P$  is true of the value of  $Z$  in  $J$  and  $V$ . Thus,  $P^Z$  "achieves" between  $J$  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 that

$$m(P^Z) = m(\text{true?} \cup x \leftarrow f(x)). \quad \blacksquare$$

Now, CF'DL is defined precisely as CFDL but using CF' instead of CF. Of course, we are interested in CFDL, not in CF'DL, but need CF'DL in which to carry out our reasoning. 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. Note that we could have defined CF' simply by adding  $P^Z$  constructs to the set of basic programs (i.e. besides assignments and tests), and then defining CF' to be the set of closed terms of width 1. However, we want to outlaw the possibility of  $P^Z$  appearing in  $\tau(X)$ , and then being " $*$ -ed", i.e. we do not want programs of the form  $\tau^*(f)$  in which  $\tau$  includes an "achieve" program. The reason for this will become apparent in the proof of Lemma 4.6.

## 4.2 Results.

*Theorem 4.1:* For any arithmetical universe  $A$ ,  $L$  is  $A$ -expressive for CF'DL.

*Proof:* The Theorem is proved similarly to Theorem 3.2, but 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  $A$  (described in Section 3.1), that there exists, for every term  $\tau(X)$ , an L-wff  $\text{ITR}_\tau(n)$  such that for every  $n$   $\text{ITR}_\tau(n)$  "expresses"  $\tau^n(\text{false?})$ , in the sense that  $m(\text{ITR}_\tau(n)^Z) = m(\tau^n(\text{false?}))$ , where  $Z = \text{var}(\tau)$ . As in Theorem 3.2, if  $Q_L$  is an arithmetical equivalent of  $Q$  then an arithmetical equivalent of  $\langle \tau^*(f) \rangle Q$  is  $\exists n \exists Z' ( \text{nat}(n) \wedge \text{ITR}_\tau(n) \wedge (Q_L)^Z )$ .  $\blacksquare$

We now show that in fact RC is embedded in CF.

*Lemma 4.2:* For every  $\alpha \in \text{CF}'$ ,

$$m(\alpha^*) = m((\text{true?} \cup \alpha; X)^*(f)) = m((\text{true?} \cup X; \alpha)^*(f)).$$

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

A counter example to the other direction of the fact implied by Lemma 4.2 is the following program  $\alpha \in CF$ , for which it can be easily shown that there does not exist any  $\beta \in RC$  such that  $m(\alpha) = m(\beta)$ :

$$(true? \cup (x \leftarrow f(x); X; x \leftarrow g(x)))^*(f).$$

Thus, CFDL falls between DL and r.e.-DL (see Section 2.3.5). Consequently, Theorems 2.8-2.11 are true of CFDL. It would be interesting, in line with the open problems of Chapter 2, to know the answers to the following:

*Open Problem:* Is  $DL < CFDL$ ?

*Open Problem:* Is  $CFDL < r.e.-DL$ ?

Note the analogy between  $\alpha^*$  and  $\tau^*(f)$ , which can be clearly seen by relaxing notation and writing

$$\begin{aligned} \alpha^* &= \bigcup_{n=0}^{\infty} \alpha^n & \tau^*(f) &= \bigcup_{n=0}^{\infty} \tau^n(\text{false?}), \\ [\alpha^*]P &= \forall n[\alpha^n]P & [\tau^*(f)]P &= \forall n[\tau^n(\text{false?})]P, \\ \langle \alpha^* \rangle P &= \exists n \langle \alpha^n \rangle P & \langle \tau^*(f) \rangle P &= \exists n \langle \tau^n(\text{false?}) \rangle P. \end{aligned}$$

In the sequel we will write  $Z=Z'$  to abbreviate  $\bigwedge_{x \in Z} (x=x')$ , and will assume that for programs of the form  $P^Z$ ,  $Z$  and  $Z'$  appear in that order in the parenthesised list of free variables of  $P$ . Thus for example,  $P(Z'', Z')$  will abbreviate  $P^Z_{Z''}$ . Furthermore, we will assume that in the context of a given universe  $U$ , the elements of  $Z$ ,  $Z''$  etc. consist of *uninterpreted* variables.

We now show how to express the fact that  $P^Z$  is an upper or lower bound on the relation represented by a program  $\alpha$ , using DL notions.

*Theorem 4.3:* For any universe  $U$  and  $\alpha \in CF$ , if  $Z = \text{var}(\alpha)$  then

$$\begin{aligned} (1) \quad \models_U (Z'=Z \supset [\alpha]P(Z', Z)) & \text{ iff } m(\alpha) \subseteq m(P^Z), \\ \text{and} \quad (2) \quad \models_U (P(Z, Z') \supset \langle \alpha \rangle Z'=Z) & \text{ iff } m(P^Z) \subseteq m(\alpha). \end{aligned}$$

*Proof:* (1): Assume  $\models_U (Z'=Z \supset [\alpha]P(Z', Z))$  and assume  $(J, f) \in m(\alpha)$ . We have to show that  $f = [V / Z]$  for some vector  $V$  of elements of  $D_J$ , and that  $[Z_f / Z'] \models P(Z, Z')$ . The

first is trivial by the fact that  $Z = \alpha x(\alpha)$ . Now, by the definition of  $m(\alpha)$ , and since  $\alpha$  does not change  $Z$ , if  $(J, f) \in m(\alpha)$  then also  $(J, f) \in m(\alpha)$ , where  $J = [Z_g / Z']$  and  $f = [Z_g / Z'] f = [Z_g / Z' X Z_g / Z]$ . However, by the assumption, since we have constructed  $J'$  such that  $J' \models (Z=Z')$ , we must have  $J' \models P(Z, Z)$ , or  $[Z_g / Z' X Z_g / Z] \models P(Z, Z)$ , which is the same as saying  $[Z_g / Z'] \models P(Z, Z)$ .

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

(2): Assume  $\vdash_U (P(Z, Z') \supset \langle \alpha \rangle Z=Z')$ , and assume  $(J, f) \in m(P^Z)$ . We prove  $(J, f) \in \alpha$ . By the second assumption,  $[Z_g / Z'] \models P(Z, Z')$ , so that by the first we have  $([Z_g, Z'] f, [Z_g / Z' X Z_g / Z] f) \in m(\alpha)$ . Thus, we can conclude that  $(J, [Z_g, Z'] f) \in m(\alpha)$ . Finally, from  $f = [V / Z] f$  for some  $V$ , we conclude that  $f = [Z_g, Z] f$ , and hence that  $(J, f) \in m(\alpha)$ .

Conversely, assume  $m(P^Z) \subseteq m(\alpha)$ , and that for some  $J \in U$ ,  $J \models P(Z, Z')$ . We show the existence of  $f \in U$  such that  $(J, f) \in m(\alpha)$  and  $Z' f = Z_g$ . Take  $f$  to be  $[Z_g / Z]$ .

Certainly  $Z' f = Z_g$ . Furthermore, by the definition of  $P^Z$ , since  $[Z_g / Z]$  is simply  $f$  itself, and since we assumed that  $J \models P(Z, Z')$ , we conclude that  $(J, f) \in m(P^Z)$ , and hence  $(J, f) \in m(\alpha)$ . ■

We note that remarks to the extent that Theorem 4.3(1) holds are implicit in various places in the literature, and in particular we mention the work on "inclusion correctness" in [3].

We now present some results, all derived from well known properties of binary relations, functionals and least fixpoints. However, since we will use them in the next section to construct our axiom system we state them in terms of relations of the form  $m(\alpha)$  for some  $\alpha \in CF$ .

**Lemma 4.4:** For any  $\alpha, \alpha' \in CF$  and term  $\tau(X)$ , if  $m(\alpha) \subseteq m(\alpha')$  then  $m(\tau(\alpha)) \subseteq m(\tau(\alpha'))$ .

*Proof:* This is the monotonicity of our  $\tau$ 's over the domain of binary relations, and we omit the standard proof. ■

**Lemma 4.5 (Park [S1]):** For any  $\alpha \in CF'$  and term  $\tau(X)$ , if  $m(\tau(\alpha)) \subseteq m(\alpha)$  then  $m(\tau^*(f)) \subseteq m(\alpha)$ .

This is Park's [S1] Fixpoint Induction Principle.

**Lemma 4.6:** For every  $\alpha_0, \alpha_1, \dots \in CF'$ , and term  $\tau(X)$ , if  $m(\alpha_0) = \emptyset$  and if furthermore for all  $i \geq 0$  we have  $m(\alpha_{i+1}) \subseteq m(\tau(\alpha_i))$ , then for all  $i \geq 0$ ,  $m(\alpha_i) \subseteq m(\tau^*(f))$ .

*Proof:* By induction on  $i$ . For  $i=0$  we have  $m(\alpha_0) \subseteq m(\tau(\alpha_0)) = m(\tau(\text{false?})) \subseteq (\bigcup_{n=0}^{\infty} m(\tau^n(\text{false?}))) = m(\tau^*(f))$ . Assume  $m(\alpha_i) \subseteq m(\tau^*(f))$ , so that by Lemma 4.4  $m(\tau(\alpha_i)) \subseteq m(\tau(\tau^*(f)))$ . Thus we have  $m(\alpha_{i+1}) \subseteq m(\tau(\alpha_i)) \subseteq m(\tau(\tau^*(f)))$ . However, one can show by induction on the structure of  $\tau$  that  $m(\tau(\bigcup_{n=0}^{\infty} \tau^n(\text{false?}))) = \bigcup_{n=0}^{\infty} m(\tau(\tau^n(\text{false?})))$ . (This follows from the continuity of  $\tau$  over the domain of binary relations; cf. [5]. We note that this would not have been true in general if  $CF'$  would have allowed achieve programs of the form  $P^Z$  to appear in the terms.) And so we have  $m(\alpha_{i+1}) \subseteq \bigcup_{n=0}^{\infty} m(\tau^n(\text{false?})) = m(\tau^*(f))$ . ■

### 4.3 Axiomatization of CF'DL.

In this section we present an arithmetically complete axiom system  $R$  for proving the A-valid CF'DL-wffs; as a corollary, of course,  $R$  is arithmetically complete for CF'DL too. In the sequel then,  $A$  is any arithmetical universe, and we adopt the same conventions regarding formulae with appearances of  $n, m, \dots$  as in Section 3.2. Also, the "achieve" program corresponding to the L-wff  $P(n, Z, Z')$  will be denoted by  $P(n)^Z$ .

Consider now the following axiom system  $R$  for CF'DL.

*Axioms:*

(A)-(F) from  $P$ ,

(K)  $[P^Z]Q \equiv (YZ'')(P(Z, Z'') \supset Q^Z)$  for L-wffs  $P$  and  $Q$ ;

(L)  $(P \supset [\tau^*(f)]Q) \supset ((P \wedge R) \supset [\tau^*(f)](Q \wedge R))$  where  $\text{var}(R) \cap \text{var}(\tau) = \emptyset$ ,

*Inference Rules:*

(G) and (H) from  $P$ ,



$$(M) \quad \frac{Z'=Z \supset [\tau(P^Z)]P(Z,Z)}{\quad}$$

where  $Z \in \text{var}(\tau)$ ,

$$Z'=Z \supset [\tau^*(f)]P(Z,Z)$$

$$(N) \quad \frac{P(n+1, Z, Z) \supset \langle \tau(P(n)^Z) \rangle Z=Z \quad \neg P(0, Z, Z)}{\quad}$$

for an L-wff P

$$P(n, Z, Z) \supset \langle \tau^*(f) \rangle Z=Z$$

where  $Z \in \text{var}(\tau)$ ,  $n \notin \text{var}(\tau)$ ,

Provability in  $R$  is defined as usual. The intuition behind axiom (I) is that it allows "carrying"  $R$  across a program when that program cannot affect the truth of  $R$ . We now establish the soundness of the additional axioms and inference rules.

**Lemma 4.7:** For any L-wffs  $T$  and  $P(Z, Z)$ , CFDL-wffs  $Q$ ,  $R$  and  $S$ , term  $\tau(X)$ , the following are valid

$$(1) \quad \vdash_{\mathcal{U}} \tau^*(f) \supset (\forall Z')(P(Z, Z) \supset T^Z),$$

$$(2) \quad (S \supset [\tau^*(f)]Q) \supset ((SAR) \supset [\tau^*(f)](QAR)), \quad \text{where } \text{var}(R) \cap \text{var}(\tau) = \emptyset$$

*Proof:* Straightforward from the definitions. ■

**Lemma 4.8:** For any universe  $\mathcal{U}$ , L-wff  $P(Z, Z)$  and term  $\tau$ , where  $Z \in \text{var}(\tau)$ , if  $\vdash_{\mathcal{U}} (Z'=Z \supset [\tau(P^Z)]P(Z, Z))$ , then  $\vdash_{\mathcal{U}} (Z=Z \supset [\tau^*(f)]P(Z, Z))$ .

*Proof:* By Theorem 4.3(1) the hypothesis is simply  $m(P^Z) \subseteq m(P^Z)$ . By Park's principle (Lemma 4.5) we obtain  $m(\tau^*(f)) \subseteq m(P^Z)$ , which, again by Theorem 4.3(1), is precisely the conclusion. ■

**Lemma 4.9:** For any L-wff  $P(n, Z, Z)$  and term  $\tau$ , where  $n \notin \text{var}(\tau)$  and  $Z \in \text{var}(\tau)$ , if

$$\vdash_A \neg P(0, Z, Z) \quad \text{and} \quad \vdash_A (P(n+1, Z, Z) \supset \langle \tau(P(n)^Z) \rangle Z=Z), \quad \text{then}$$

$$\vdash_A (P(n, Z, Z) \supset \langle \tau^*(f) \rangle Z=Z).$$

*Proof:* One can show that  $\vdash_A \neg P(0, Z, Z)$  is in fact equivalent to saying that  $m(P(0)^Z) = \emptyset$ . Furthermore, by Theorem 4.3(2) the second assumption amounts to asserting that  $m(P(n+1)^Z) \subseteq m(\tau(P(n)^Z))$ . By Lemma 4.5 we conclude that  $m(P(n)^Z) \subseteq m(\tau^*(f))$  for all  $n$ . Thus, again by Theorem 4.3(2), we have the conclusion. ■

**Theorem 4.10 (A-soundness of R):** For any CF'DL-wff P, if  $\vdash_R P$  then  $\vdash_A P$ .

*Proof:* Follows from Theorem 3.6 and Lemmas 4.7, 4.8 and 4.9. ■

Again we will apply Theorem 3.1 to prove the arithmetical completeness of R, but we are required first to prove the appropriate Box- and Diamond-completeness theorems. These will be established with the aid of:

**Lemma 4.11:** The following are derived rules of R where Z and n are as in (M) and (N):

$$(M') \quad Z'=Z \supset [\tau(P^Z)]P(Z',Z) \quad , \quad R \supset [P^Z]Q$$

---


$$R \supset [\tau^*(f)]Q$$

$$(N') \quad P(n+1, Z, Z') \supset \langle \tau(P(n)^Z) \rangle Z=Z' \quad , \quad \neg P(0, Z, Z') \quad , \quad R \supset \exists n \langle P(n)^Z \rangle Q$$

---


$$R \supset \langle \tau^*(f) \rangle Q$$

*Proof:* (M'): Assume  $\vdash_R (Z'=Z \supset [\tau(P^Z)]P(Z',Z))$ . We apply (M) to obtain  $\vdash_R (Z'=Z \supset [\tau^*(f)]P(Z',Z))$ . Using axiom (L) we get  $\vdash_R ((Z=Z \wedge (\forall Z'') (P(Z',Z) \supset Q_{Z'}^{Z''})) \supset [\tau^*(f)](P(Z',Z) \wedge (\forall Z'') (P(Z',Z'') \supset Q_{Z'}^{Z''})))$ , from which we deduce  $\vdash_R ((\forall Z'') (P(Z',Z'') \supset Q_{Z'}^{Z''}) \supset [\tau^*(f)]Q)$ . Thus, by axiom (K) and the second assumption the conclusion follows.

(N'): Similar to (M'). ■

Note the similarity between rules (I') and (J') of Lemma 3.7 on one hand, and (M') and (N') on the other. Here too, for the  $[\tau^*(f)]$  case we are essentially looking for what we might call an "invariant" P under the application of  $\tau$ , which is "between" R and Q in the sense of  $R \supset [P^Z]Q$ . For the  $\langle \tau^*(f) \rangle$  case we are "counting" the number of applications of  $\tau$  needed for being able to terminate in a state satisfying Q.

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

**Lemma 4.12 (Invariance Lemma for CF'DL):** For every term  $\tau(X)$  and CF'DL-wffs R and Q, if  $\vdash_A (R \supset [\tau^*(f)]Q)$  then there exists an L-wff  $P(Z, Z')$  with  $Z = \text{var}(\tau)$ , such that

$$\vdash_A (R \supset [P^Z]Q) \quad \text{and} \quad \vdash_A (Z'=Z \supset [\tau(P^Z)]P(Z',Z)).$$

*Proof:* Implied by the way Theorem 4.1 is proved is the fact that there exists a first order formula of arithmetic  $P(Z, Z')$  which "represents" the program  $\tau^*(f)$  in the sense that  $m(P^Z) = m(\tau^*(f))$ . Certainly then, by the assumption, we have  $\vdash_A (R \supset [P^Z]Q)$ . Also, as noted in the proof of Lemma 4.6,  $m(\tau(\tau^*(f))) = m(\tau^*(f))$ , and so we have  $m(\tau(P^Z)) \subseteq m(P^Z)$ , which by Theorem 4.3(1) is  $\vdash_A (Z'=Z \supset [\tau(P^Z)]P(Z, Z'))$ . ■

**Theorem 4.13 (Box-completeness Theorem for CF'DL):** For every  $\alpha \in CF'$  and L-wffs  $R$  and  $Q$ , if  $\vdash_A (R \supset [\alpha]Q)$  then  $\vdash_R (R \supset [\alpha]Q)$ .

*Proof:* The proof follows Theorem 3.9 precisely, but uses Lemma 4.12 and rule (M') instead of Lemma 3.8 and rule (I'). ■

**Lemma 4.14 (Convergence Lemma for CF'DL):** For every term  $\tau(X)$  and CF'DL-wffs  $R$  and  $Q$ , if  $\vdash_A (R \supset \langle \tau^*(f) \rangle Q)$  then there exists an L-wff  $P(n, Z, Z')$  such that  $\vdash_A (P(n+1, Z, Z') \supset \langle \tau(P(n)^Z) \rangle Z=Z')$ ,  $\vdash_A \neg P(0, Z, Z')$ , and  $\vdash_A (R \supset \exists n \langle P(n)^Z \rangle Q)$ .

*Proof:* Again, by the method used in the proof of Theorem 4.1, there exists an L-wff  $P(n, Z, Z')$  representing  $\tau^n(\text{false?})$  in the sense that for every  $n$  we have  $m(P(n)^Z) = m(\tau^n(\text{false?}))$ . It is easy to see that all three A-validities hold for  $P$ . ■

**Theorem 4.15 (Diamond-completeness Theorem for CF'DL):** For every  $\alpha \in CF'$  and L-wffs  $R$  and  $Q$ , if  $\vdash_A (R \supset \langle \alpha \rangle Q)$  then  $\vdash_R (R \supset \langle \alpha \rangle Q)$ .

*Proof:* Precisely as Theorem 3.11, but using Lemma 4.14 and rule (N') instead of Lemma 3.10 and rule (J'). ■

Here too we conclude that for CF'DL-wffs, A-validity and provability in  $R$  are equivalent concepts:

**Theorem 4.16 (Arithmetical Soundness and Completeness for CF'DL):** For every CF'DL-wff  $P$ ,  $\vdash_A P$  iff  $\vdash_R P$ .

*Proof:* One direction is Theorem 4.10, and the other follows from Theorems 3.1, 4.1, 4.13 and 4.15. ■

We remark that the  $[a]$  part of  $R$ , in particular the derived rule ( $M'$ ), conveys the essential ideas appearing in the axiom systems of [19] and [23] for proving the partial correctness of recursive programs. We have essentially shown that the central idea in these axiomatizations (referred to in [23] as the "freezing of the variables" method) is in fact a rephrasing of Park's [51] induction principle in a logical framework. Rule (N) for  $\langle \tau^*(f) \rangle$  is very similar to the rule in [63] for proving the total correctness of deterministic recursive programs.

The results in this section indicate that reasoning about "pure" recursion is analogous to that of reasoning about regular ones. Here we are using the integers to count how "deep" we are in the recursion (using  $P(n)^Z$ ), whereas for  $\alpha^*$  we counted how "far" we are in the iteration. Other than having to devise the  $P^Z$  machinery, there was no real difficulty at this point in extending the methods of Chapter 3 to recursive programs. In Chapter 7, though, a reassessment of this claim will become necessary.

An interesting remark, which we do not elaborate upon nor justify farther here, is the fact that the proof method for formulae of the form  $R \rightarrow [a]Q$  which is incorporated into  $R$  boils down to Floyd's [17] *inductive assertion* method and to Morris and Wegbreit's [45] *subgoal induction* method respectively, when regular programs are translated into recursive ones via the two methods appearing in Lemma 4.1. Thus the quality holding between these two methods shows up nicely as stemming from two dual ways of viewing  $\alpha^*$ .

#### 4.4 Mutual Recursion.

In this section we briefly indicate how to extend the axiomatization of Section 4.3 to the case where the programs can be *mutually recursive*. Specifically, we consider the programming language MCF (giving rise to the logic MCF'DL), which is the set of all simple closed terms, i.e.  $T = \bigcup_{j=0}^{\infty} T_j$

We do not provide here a precise definition of  $m(\mu_1 X_1 \dots X_n (\tau_1, \dots, \tau_n))$ , but rather assume that the reader is familiar with the standard definition of it (cf. [3] or [26]) as the  $i$ 'th component of the *least* solution of the system of relational equations

$$\begin{cases} X_1 = \tau_1(X_1, \dots, X_n) \\ \vdots \\ X_n = \tau_n(X_1, \dots, X_n), \end{cases}$$

where the ordering on the binary relations is that of set inclusion.

The axiom system **MR** for **MCF\*DL** is constructed analogously to **R**. Axiom **(L)** is rephrased for a general  $\mu$ -term as

$$(L') \quad (P \supset [\mu_j X_1 \dots X_n (\tau_1, \dots, \tau_n)] Q) \supset ((P \wedge R) \supset [\mu_j X_1 \dots X_n (\tau_1, \dots, \tau_n)] (Q \wedge R)),$$

where  $\text{var}(R) \cap \text{var}(\tau_1 \cup \dots \cup \tau_n) = \emptyset$ .

Denote by  $\mu_j(i, \alpha)$  the program

$$\mu_j X_1 \dots X_{i-1} X_{i+1} \dots X_n (\tau_1(X_1, \dots, X_{i-1}, \alpha, X_{i+1}, \dots, X_n), \dots, \tau_n(X_1, \dots, X_{i-1}, \alpha, X_{i+1}, \dots, X_n)).$$

$\mu_j(i, \alpha)$  is the program  $\mu_j X_1 \dots X_n (\tau_1, \dots, \tau_n)$  in which the  $i$ th "procedure" has been replaced by the program  $\alpha$ ; wherever a "call" is made to this procedure, in which case  $\tau_i$  is to be executed,  $\alpha$  is executed instead.

The rules for the recursive constructs are

$$(M') \quad Z' = Z \supset [\tau_1(\mu_1(i, P^Z), \dots, \mu_{i-1}(i, P^Z), P^Z, \mu_{i+1}(i, P^Z), \dots, \mu_n(i, P^Z))] P(Z, Z)$$

---


$$Z' = Z \supset [\mu_j X_1 \dots X_n (\tau_1(X_1, \dots, X_n), \dots, \tau_n(X_1, \dots, X_n))] P(Z, Z)$$

where  $Z = \text{var}(\tau_1 \cup \dots \cup \tau_n)$ ,

(N')

$$P(n+1, Z, Z') \supset \langle \tau_1(\mu_1(i, P(n)Z)), \dots, \mu_{i-1}(i, P(n)Z), P(n)Z, \mu_{i+1}(i, P(n)Z), \dots, \mu_n(i, P(n)Z) \rangle Z=Z', \\ \neg P(0, Z, Z')$$

$$P(n, Z, Z') \supset \langle \mu_1(X_1 \dots X_n), \tau_1(X_1 \dots X_n), \dots, \tau_n(X_1 \dots X_n) \rangle Z=Z'$$

where  $Z = \text{var}(\tau_1 \cup \dots \cup \tau_n)$  and  $n \neq \text{var}(\tau_1 \cup \dots \cup \tau_n)$ .

It should be noted that the premises of both these rules involve programs of less complexity than their conclusions; the latter involve terms in  $T_n$ , whereas the premises involve "at most" terms in  $T_{n-1}$ .

One can now show the following, by a detailed argument analogous to that followed in Sections 4.2 and 4.3:

**Theorem 4.17:** For every MCFDL-wff  $P$ ,  $\vdash_A P$  iff  $\vdash_{MFL} P$ .

We remark that rule (N') essentially follows our ideas in [14] for proving the partial correctness of recursive programs.

## PART II: Computation-Tree Based Logics.

### 5. Computation Trees, Total Correctness and Weakest Preconditions.

Up to this point we have been developing mathematical tools, namely the various dynamic logics, which enabled us to write down and prove certain formulae which made assertions about programs. In Section 2.2 we commented to the extent that some conventional properties of programs which have intuitively plausible meanings, happen to be expressible as simple formulae of dynamic logic.

In this chapter we show that an important property of a program, namely its so called "total correctness", does not have a straightforward intuitive meaning, and that its definition requires careful analysis of the notion of "executing" a program. In fact, the definition of the total correctness of a program *depends* upon the particular method of execution one has in mind. Consequently, it is not at all clear *a priori* whether this property of a program can be expressed in dynamic logic. An upshot is the fact that the closely related notion of the *weakest precondition* (*wp*) of a program, although introduced by Dijkstra in [13] and used extensively in the literature, has not received a proper definition in [13] or in [14]. The objective of this chapter is to clarify, and to precisely define, both of these concepts.

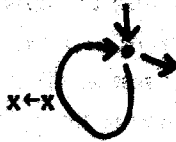
In Section 5.1 we motivate and introduce the problem. Section 5.2 contains a refinement of the binary relation semantics for our programming language  $RC$ , using *computation trees*, and giving rise to the two important concepts of *diverging* and *failing*. In Section 5.3 we introduce four plausible methods for executing nondeterministic programs, by describing four methods for traversing computation trees in search of a final state. The *total correctness* of a program is then defined as being dependent upon these methods. In Section 5.4 we use these ideas to define the corresponding weakest precondition which similarly depends on execution methods, and to analyze each of the four resulting *wp*'s as to whether they satisfy the properties required of Dijkstra's *wp* in [14]. We find that two of them do. Then, in Section 5.5, we define the *guarded commands* language introduced in [13], and carry out a formal analysis aimed at showing that Dijkstra really had in mind *one* particular notion of *wp*, which corresponds only to one of our four execution methods, namely depth-first search without backtracking.

## 5.1 Motivation.

Let us look at two examples.

(1) It is easy to see that any DL-wff  $P(\alpha)$  involving the program variable  $\alpha$  has the property that  $P(\beta)$  is equivalent to  $P(\beta')$  in every state, where  $\beta$  is taken to be  $(x \leftarrow e)$  and  $\beta'$  to be  $(x \leftarrow e \cup (y \leftarrow e'; \text{false?}))$ . This is simply because  $m(\beta) = m(\beta')$ . However, we would like to be able to state that if  $\beta'$  is "executed" by the processor "choosing" one of the components of the  $\cup$  connective and executing it, then if  $(y \leftarrow e'; \text{false?})$  happens to be chosen this "execution" will not terminate.

(2) Similarly,  $P(\gamma)$  is always equivalent to  $P(\gamma')$ , where  $\gamma$  is taken to be  $(x \leftarrow e)$  and  $\gamma'$  to be  $(x \leftarrow e; (x \leftarrow x)^*)$ . Here too  $m(\gamma) = m(\gamma')$ , but we would like to be able to state that if  $(x \leftarrow x)^*$  is executed by the reasonable method of, repeatedly, at each step either terminating or executing  $x \leftarrow x$ , as suggested by the diagram



then there is a possibility of never choosing to terminate and hence executing  $x \leftarrow x$  "for ever".

We would like to refer to the phenomenon illustrated by example (1) as a *failure* and to that illustrated by (2) as a *divergence*.

Intuitively, a failure indicates reaching a false test with no immediate alternative at hand. In example (1) above, in order to carry out the alternative  $x \leftarrow e$  when the false test is reached the assignment  $y \leftarrow e'$  must be "undone"; thus the alternative entails some *backtracking* and is therefore not immediate; consequently, there is a failure. However, the *if P then  $\alpha$  else  $\beta$*  construct which is  $(P?; \alpha \cup \neg P?; \beta)$  (see Section 2.3.4), should not contain a failure although one of the tests will be false whenever the construct is reached; here there is an immediate alternative at hand. A divergence is what is more popularly called an "infinite loop"; i.e. a computation that does not terminate. These two concepts will receive formal definitions in the next section.

What we are interested in defining, is a precise notion of the *total correctness* of a program  $\alpha$ , with respect to assertions  $R$  and  $Q$ , to mean intuitively that whenever  $R$  is



true, then "no matter how  $\alpha$  is executed" (i.e. "no matter how choices are made") it is the case that  $\alpha$  will indeed terminate in a state satisfying  $Q$ . It might seem plausible at this point that we would want this definition to be such that  $\beta$  and  $\gamma$  are, but  $\beta'$  and  $\gamma'$  are *not*, totally correct with respect to *true* and *true*. In other words, it might seem that the possibility of either diverging or failing should render a program not totally correct. We will see in Section 5.3 that this is *not* the case. In fact, we will show that the four possibilities obtained by having the presence of a divergence / failure affect / not-affect the total correctness of a program, correspond smoothly to four different methods of execution of nondeterministic programs.

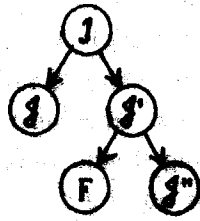
We now set up the technical machinery we need.

## 5.2 Computation Trees, Diverging and Failing.

In this section we introduce the notion of the *J-computation tree* of a program  $\alpha$ , denoted by  $ct(\alpha, J)$ . We present some properties of computation trees and in particular show that one might view computation trees as an alternative semantics for the set of regular programs  $RG$ , consistent with the binary relation semantics. The trees however, in addition to the input-output information, contain much more; for example, they contain information regarding the presence or absence of divergences and failures.

Each node of  $ct(\alpha, J)$  will be labeled with a state in  $\Gamma$  or with the symbol  $F$  (denoting failure), and will be of outdegree at most 2. The root is labeled with  $J$  and nodes labeled with  $F$  will always be leaves. The intuition is that a path from the root represents a legal computation of  $\alpha$  starting in state  $J$ . Accordingly, a leaf represents a termination state if it is labeled with a state in  $\Gamma$ , or a failure if it is labeled with  $F$ . Any node with descendants represents an intermediate state of  $\alpha$ . 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  $(t, l)$ , 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  $l$  (the label of the node) is either a state in  $\Gamma$  or the symbol  $F$ . Thus, for example, the tree



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

In order to define  $ct(\alpha, J)$  we first define a preliminary tree  $pct(\alpha, J)$  in which every false test will be indicated by a failure node.  $ct(\alpha, J)$  will then be obtained from  $pct(\alpha, J)$  by deleting those failure nodes for which there is an immediate non-failure alternative.

Formally, for any  $J \in \Gamma$  and  $\alpha \in RC$ , we define, by induction on the structure of  $\alpha$ , the preliminary computation tree  $pct(\alpha, J)$  to be a subset of  $\{0, 1\}^* \times (\Gamma \cup \{F\})$  as follows, where we use  $l$  to range over  $(\Gamma \cup \{F\})$ , and  $s, t, \dots$  to range over  $\{0, 1\}^*$ :



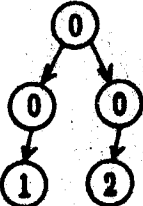
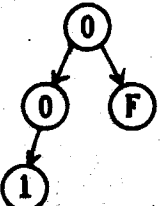

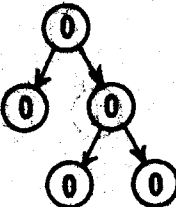
- (1)  $pct(x \leftarrow e, J) = \{(\lambda, J), (0, [e_j / x]J)\}$ ,
- (2)  $pct(P?, J) = \begin{cases} \{(\lambda, J)\} & \text{if } J \models P \\ \{(\lambda, F)\} & \text{if } J \not\models P, \end{cases}$
- (3)  $pct(\alpha \cup \beta, J) = \{(\lambda, J)\} \cup \{(0t, l) \mid (t, l) \in pct(\alpha, J)\} \cup \{(1t, l) \mid (t, l) \in pct(\beta, J)\}$ ,
- (4) Let  $E = \{(t, J) \in pct(\alpha, J) \mid J \in \Gamma \wedge (\forall b \in \{0, 1\}) (\forall l \in (\Gamma \cup \{F\})) ((tb, l) \notin pct(\alpha, J))\}$ ,  
and let  $G = pct(\alpha, J) - E$ . Then  
 $pct(\alpha; \beta, J) = G \cup \{(ts, l) \mid (\exists J) ((t, J) \in E \wedge (s, l) \in pct(\beta, J))\}$ ,
- (5)  $pct(\alpha^*, J) = pct((true? \cup \alpha; \alpha^*), J)$ .

Note that clause (5) might give rise to an infinite tree.

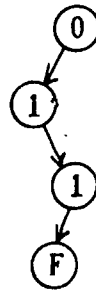
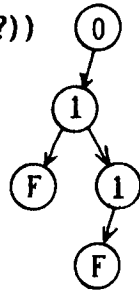
Now obtain  $ct(\alpha, J)$  from  $pct(\alpha, J)$  by deleting some of the failure nodes as follows: for any  $t \in \{0, 1\}^*$  and  $J \in \Gamma$ , replace every pair in  $pct(\alpha, J)$  of the form  $(t0, F)$ ,  $(t1, J)$  by

$(t1, \mathcal{J})$ , and of the form  $(t1, F)$ ,  $(t0, \mathcal{J})$  by  $(t0, \mathcal{J})$ . Thus we are ignoring false tests which occur as a component of the  $\cup$  operator, when the other component is not a false test.

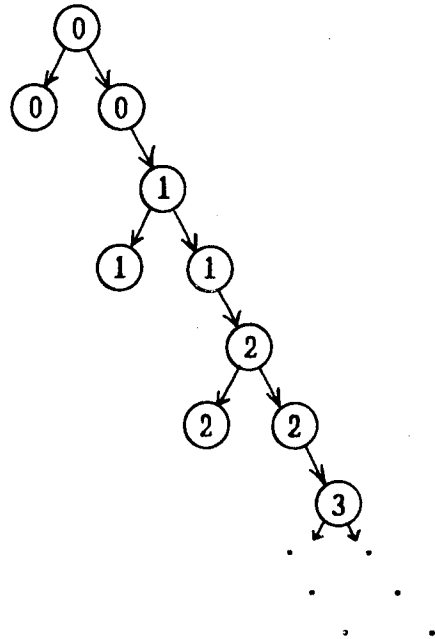
*Examples:* We describe by means of simple diagrams, some computation trees for various  $\alpha$ . In each case, whenever they are not identical, we give both the preliminary tree  $pct(\alpha, \mathcal{J})$  and the final tree  $ct(\alpha, \mathcal{J})$ . In all the examples  $\mathcal{J}$  is some fixed state of the arithmetical universe  $N$ , for which  $x_{\mathcal{J}}=0$ , and in the diagrams we let  $t$  denote the state  $[t/x]\mathcal{J}$ .

$\alpha$	$pct(\alpha, \mathcal{J})$	$ct(\alpha, \mathcal{J})$
$x=0?$		
$x=1?$		
$x=0?; x \leftarrow x+1 \cup x < 2?; x \leftarrow x+2$		
$x=0?; x \leftarrow x+1 \cup x \neq 0?; x \leftarrow x+2$		
$x=0? \cup (x=0? \cup x=0?)$		

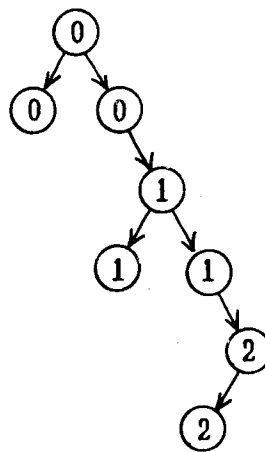
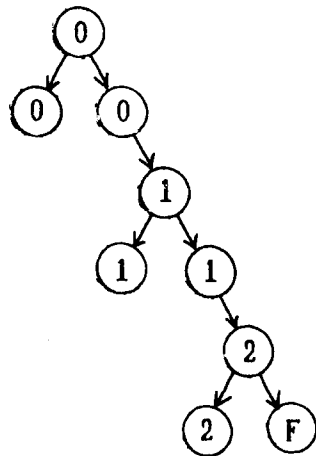
$x \leftarrow x+1; (x=0? \cup (x \leftarrow x+1; x=1?))$



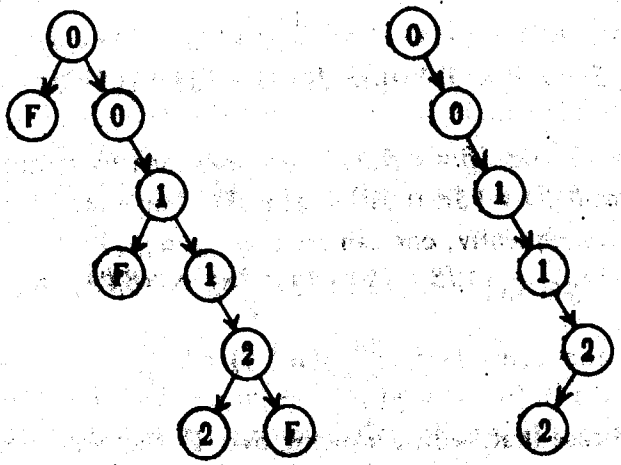
$(x \leftarrow x+1)^*$



$(x < 2?; x \leftarrow x+1)^*$



$(x < 2?; x \leftarrow x+1)^*; x = 2?$



Thus, the computation tree  $cr(\alpha, J)$  of statements of the form *if P then  $\beta$  else  $\gamma$*  and *while P do  $\beta$*  (see Section 2.3A) do not contain failures other than the relevant ones inside  $\beta$  or  $\gamma$ .

- Lemma 5.1:** For every  $\alpha \in RC$ ,  $l \in (\Gamma \cup \{F\})$  and  $J \in \Gamma$ ,
- (1) there is a unique node  $(\lambda, J')$  in  $cr(\alpha, J)$  for some  $J' \in (\Gamma \cup \{F\})$ ,
  - (2) for every  $t \in \{0, 1\}^*$  there is at most one node in  $cr(\alpha, J)$  of the form  $(t, l)$ ,
  - (3) for every  $t \in \{0, 1\}^*$  and  $b \in \{0, 1\}$ , if  $(tb, l) \in cr(\alpha, J)$ , then  $(t, J) \in cr(\alpha, J)$  for some  $J \in \Gamma$ .

*Proof:* Omitted. ■

Thus, for every  $\alpha$  and  $J$ ,  $cr(\alpha, J)$  is a nonempty, possibly infinite tree of finite outdegree with nodes labeled with elements of  $\Gamma \cup \{F\}$ . Nodes of the form  $(t, l)$  and  $(t, l)$  are called *descendants* of a node of the form  $(t, J)$ , and a node with no descendants is called a *leaf*. By Lemma 5.1(2,3) all nodes labeled with  $F$  are leaves.

We now show that computation trees subsume the binary relation semantics of Chapter 2.

**Theorem 5.2:** For any  $\alpha \in RC$ ,  $(J, J) \in m(\alpha)$  iff  $cr(\alpha, J)$  has a leaf labeled with  $J$ .

*Proof:* By induction on the structure of  $\alpha$ . Denote by  $J\alpha$  the set  $\{J \mid J\alpha J\}$ , and by  $s(\alpha, J)$  the set  $\{J \mid \text{there is a leaf of } cr(\alpha, J) \text{ labeled with } J\}$ . We prove first that  $J\alpha = s(\alpha, J)$ , and then the result follows by observing that the transition from  $pcr(\alpha, J)$  to  $cr(\alpha, J)$  does not delete any nodes which contribute to the set  $s(\alpha, J)$ .

For an assignment, we have  $J(x \leftarrow e) = \{[e_j / x]J\} = s(x \leftarrow e, J)$ . For a test, if  $J \models P$  then  $J(P?) = \delta = s(P?, J)$ , and if  $J \not\models P$  then  $J(P?) = \{J\} = s(P?, J)$ .

Assume  $J\alpha = s(\alpha, J)$  and  $J\beta = s(\beta, J)$ . Certainly then by definition of  $pcr(\alpha \cup \beta, J)$  we have  $s(\alpha \cup \beta, J) = (J\alpha \cup J\beta) = J(\alpha \cup \beta)$ . For  $\alpha; \beta$ , note that in fact  $E = \{(t, J) \mid J \in s(\alpha, J)\}$ . Consequently, one can see that  $s(\alpha; \beta, J) = (\cup_{J \in s(\alpha, J)} J)\beta = (J \mid (J) (J \in J \wedge J \in J)) = J(\alpha; \beta)$ .

Similarly, one can show that  $s(\alpha^*, J) = \cup_{n=0}^{\infty} s(\alpha^n, J) = \cup_{n=0}^{\infty} J(\alpha^n) = J(\alpha^*)$ . ■

It is therefore the case that, with  $J$  ranging over  $\mathcal{D}$ , the leaves of  $cr(\alpha, J)$  which are labeled with states convey the input-output information contained in the binary relation  $m(\alpha)$ . Note that in this framework  $J \models \langle \alpha \rangle P$  asserts the existence in  $cr(\alpha, J)$  of at least one leaf labeled with a state which satisfies  $P$ . Similarly,  $J \models \langle \alpha \rangle \neg P$  asserts that  $P$  holds in any state which labels a leaf in  $cr(\alpha, J)$ . However,  $cr(\alpha, J)$  contains much more information than is contained in  $m(\alpha)$ . In particular we now define, for every program  $\alpha \in RG$ , two Boolean constants  $loop_\alpha$  and  $fail_\alpha$  which are to have the intuitive meaning of being true in state  $J$  iff  $\alpha$  can *diverge* or *fail* respectively.

Formally, we define

$$\begin{aligned} J \models loop_\alpha & \text{ iff } cr(\alpha, J) \text{ is infinite,} \\ J \models fail_\alpha & \text{ iff } cr(\alpha, J) \text{ has a node labeled with F.} \end{aligned}$$

Note that,  $cr(\alpha, J)$  being of finite outdegree, we can apply Koenig's lemma (see [31]) to conclude that in fact  $J \models loop_\alpha$  iff there exists an *infinite path* from the root; i.e. there is an infinite sequence of nodes in  $cr(\alpha, J)$

$$(\lambda, J), (b_1, J_1), (b_1 b_2, J_2), \dots, (b_1 \dots b_i, J_i), \dots$$

Hence the term "divergence".

An interesting problem is that of determining how hard it is to decide if a program diverges for "uninterpreted reasons". Formally:

*Open Problem:* What is the degree of undecidability of the set of valid formulae of the form  $P \supset loop_\alpha$ , where  $P$  is an L-wff?

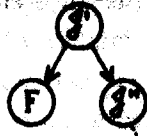
We now prove some properties of  $loop_\alpha$  and  $fail_\alpha$  which will be needed in Section 5.4. However, the main logical treatment of these concepts will be given in Chapters 6 and 7.

**Lemma 5.3:** For any  $\alpha, \beta \in RG$  the following are valid:

- (1)  $loop_{\alpha;\beta} \equiv (loop_\alpha \vee \langle \alpha \rangle loop_\beta)$ ,
- (2)  $fail_{\alpha;\beta} \supset (fail_\alpha \vee \langle \alpha \rangle fail_\beta)$ ,
- (3)  $fail_\alpha \supset fail_{\alpha;\beta}$ ,
- (4)  $\langle \alpha \rangle false \supset (fail_\alpha \vee loop_\alpha)$ .

*Proof:* (1): Assuming  $J \models loop_{\alpha;\beta}$ , consider an infinite path from the root in  $ct(\alpha;\beta, J)$ . It is easy to see that either that whole path appears in  $ct(\alpha, J)$ , or a finite initial segment of it does, and the rest (i.e. an infinite path) appears in  $ct(\beta, J)$  for some  $J' \in (J\alpha)$ . Conversely, an infinite path in either  $ct(\alpha, J)$  or in  $ct(\beta, J)$  for some  $J' \in (J\alpha)$ , will always show up in  $ct(\alpha;\beta, J)$ .

(2): Consider a failure in  $ct(\alpha;\beta, J)$ , and assume that  $J \models \neg fail_\alpha$  and  $J \models \neg fail_\beta$  for every  $J' \in (J\alpha)$ . The F-node in  $ct(\alpha;\beta, J)$  appeared in  $pct(\alpha;\beta, J)$ , and also in either  $pct(\beta, J)$  or in  $pct(\beta, J')$  for some  $J' \in (J\alpha)$ . However, for it to have been deleted in the process of constructing  $ct(\alpha, J)$  or  $ct(\beta, J')$ , it had to have appeared (wlg) in a subtree of the form



This subtree appears also in  $pct(\alpha;\beta, J)$ , and the F-node would have had to be deleted from it too.

The proofs of (3) and (4) follow similar reasoning, and are omitted. ■

Note that a counter example to the other direction of Lemma 5.3(2) is obtained by taking  $\alpha$  to be  $(true? \cup x+1)$  and  $\beta$  to be  $x=1?$ . When  $x_j=0$ , we have  $(J, J) \in m(\alpha)$  and  $J \models fail_\beta$ , but  $J \models \neg fail_{\alpha;\beta}$ .

### 5.3 Execution Methods and Total Correctness.

In this section we define four algorithms for traversing the  $J$ -computation tree  $ct(\alpha, J)$  of a program  $\alpha \in RC$  in search of a final state; i.e. a leaf of  $ct(\alpha, J)$  of the form  $(t, J)$  for some  $J \in \Gamma$ . The algorithms will output this state  $J$ . Then we define the notion of *total correctness* of a program  $\alpha$  with respect to input-output conditions  $R$  and  $Q$  as being dependent upon the methods.

We use informal terms for describing our algorithms:

*Depth Search (D)*: Starting from the root of  $t(\alpha, J)$  proceed down the tree by moving from father to son. Whenever a node with two sons is reached one of them is chosen nondeterministically and traversal continues on it. The process terminates when a leaf is reached; its label is taken as the result.

Note that if  $J \neq loop_\alpha$  holds then, using method (D), it might be the case that the particular sequence of choices made along the way will result in the traversal proceeding along an infinite path (divergence of  $\alpha$ ) and hence never terminating. Also, if  $J \neq fail_\alpha$  holds, then that sequence might result in the traversal arriving at a failure leaf and thus producing  $F$  as the result.

*Depth Search with Backtracking (DT)*: As in (D), the difference being that if a leaf labeled  $F$  is reached the procedure backtracks to the most recent choice point and tries the alternative. If that has already been tried it backtracks to the next recent one and so on. If the tree is exhausted this way execution terminates with  $F$  as the result.

Note that here too,  $J \neq loop_\alpha$  implies that the traversal might continue for ever along a divergence. However, the existence of at least one non- $F$  leaf (which can be asserted by  $J \neq \langle \alpha \rangle true$ ) guarantees that even if  $J \neq fail_\alpha$  holds the traversal will not end with  $F$  as the result.

*Breadth Search (B)*: A nonnegative integer  $k$  is chosen nondeterministically. Starting from the root the procedure moves down the tree from father to son. Whenever a node with two sons is encountered track is kept of both alternatives by working in parallel. When any leaf is encountered its label is added to an initially empty set  $RES$ . When depth  $k$  of the tree is reached, or when the tree has been exhausted,  $RES$  is checked



for emptiness. If  $RES \neq \emptyset$  the traversal terminates and an element of  $RES$  is chosen nondeterministically as the result. If  $RES = \emptyset$  and the tree has not yet been exhausted another integer  $k' > k$  is chosen nondeterministically and the procedure continues as above. Otherwise the procedure terminates with  $F$  as the result.

(Remark: the mechanism of introducing a choice of an integer  $k$  is present in order to render each leaf a possible outcome of the algorithm. A simple breadth-first search would favour higher leaves.)

Note that here if  $J \models fail_{\alpha}$  holds then the  $F$  symbol might end up being the result, as a consequence of a particular choice of  $k$  and of the element in  $RES$ . However, if at least one leaf ( $F$  or other) is present, then even if  $J \models loop_{\alpha}$  holds the procedure is guaranteed to terminate eventually because  $RES$  will become nonempty at some point.

*Breadth Search with Ignoring (BG)*: As in (B), the difference being that if an  $F$ -leaf is encountered the symbol  $F$  is not added to the set  $RES$ .

Note that here, if at least one non- $F$  leaf is present, neither can the truth of  $loop_{\alpha}$  in state  $J$  result in the procedure not halting, nor can the truth of  $fail_{\alpha}$  in  $J$  result in the procedure producing  $F$  as its result.

We remark here that the four methods presented form by no means a complete list. One can think of other methods, such as "left-first search", in which the left branch is always tried first. We feel, however, that the four we described represent the reasonable "fair" methods in which no specific group of leaves is drastically favoured over others.

We summarize the remarks that were made after each method was described as follows, where the entry 0 for a certain method under *divergence* (resp. *failure*) means that even under the assumption  $J \models \langle \alpha \rangle true$ , the fact that  $J \models loop_{\alpha}$  (resp.  $J \models fail_{\alpha}$ ) holds can result in the procedure failing to produce a final state  $\in \Gamma$  as its result:

	<i>divergence</i>	<i>failure</i>
D	0	0
DT	0	1
B	1	0
BG	1	1

We now take a close look at the sought notion of total correctness. We would like to define  $\alpha$  to be totally correct with respect to an input condition  $R$  and an output condition  $Q$  if, intuitively, starting execution of  $\alpha$  in a state in which  $R$  is true will undoubtedly result in that execution terminating in a state in which  $Q$  is true. Assume that  $J$  is a state such that  $J \models R$  holds. For  $\alpha$  to be totally correct with respect to  $R$  and  $Q$  there certainly must exist a non- $\perp$  leaf in  $ct(\alpha, J)$ ; thus we require that  $J \models \langle \alpha \rangle \text{true}$  holds. Furthermore, all such leaves are likely to be the result of any one of the four procedures described above. Thus we require in addition that every state with which such a leaf is labeled should satisfy  $Q$ ; in other words we need  $J \models \langle \alpha \rangle Q$  to hold. It is now quite evident that in order for a traversal, using one of the four methods, to be *guaranteed* to terminate with a final state as the result, we have to require that  $ct(\alpha, J)$  be free of divergences or failures if and only if a  $\theta$  appears in the corresponding column for that method in the above table.

We thus arrive at the following:

*Definition:* Given a universe  $U$ , a program  $\alpha \in RC$  and formulae  $R$  and  $Q$ , we say that  $\alpha$  is

$$\begin{aligned} \text{D-totally correct wrt } R \text{ and } Q & \text{ iff } \models_U (R \supset (\langle \alpha \rangle \text{true} \wedge \langle \alpha \rangle Q \wedge \neg \text{loop}_\alpha \wedge \neg \text{fail}_\alpha)), \\ \text{DT-totally correct wrt } R \text{ and } Q & \text{ iff } \models_U (R \supset (\langle \alpha \rangle \text{true} \wedge \langle \alpha \rangle Q \wedge \neg \text{loop}_\alpha)), \\ \text{B-totally correct wrt } R \text{ and } Q & \text{ iff } \models_U (R \supset (\langle \alpha \rangle \text{true} \wedge \langle \alpha \rangle Q \wedge \neg \text{fail}_\alpha)), \\ \text{BG-totally correct wrt } R \text{ and } Q & \text{ iff } \models_U (R \supset (\langle \alpha \rangle \text{true} \wedge \langle \alpha \rangle Q)). \end{aligned}$$

In the next section we use this definition in order to define the concept of the weakest precondition of a program with respect to an assertion, and to clarify Dijkstra's [13] notion of  $wp(\alpha, P)$ .

## 5.4 Weakest Preconditions.

The notion of the weakest precondition of a program  $\alpha$  with respect to a post condition  $Q$  was introduced by Dijkstra [13] who wrote (in [14]):

- (\*) "We shall use the notation  $wp(\alpha, Q)$  to denote the weakest precondition for the initial state of the system such that activation of  $\alpha$  is guaranteed to lead to a properly terminating activity leaving the system in a final state satisfying the post condition  $Q$ ."

Here "weakest" is in the sense that  $wp(\alpha, Q)$  is to be the largest set of states each of which has the property that "activation of  $\alpha$ " starting from that state "is guaranteed to lead to ... etc."

Other than (\*), there is no formal definition of  $wp(\alpha, Q)$  either in [13] or in [14]. However, [14] contains essentially four properties that  $wp(\alpha, Q)$  must satisfy:

- P1.  $\models (wp(\alpha, false) \equiv false)$ ,  
 P2. if  $\models (P \supset Q)$  then  $\models (wp(\alpha, P) \supset wp(\alpha, Q))$ ,  
 P3.  $\models (wp(\alpha, P \wedge Q) \equiv (wp(\alpha, P) \wedge wp(\alpha, Q)))$ ,  
 P4. (continuity): for any arithmetical universe  $A$ , if  $\models_A (\forall n)(P(n) \supset P(n+1))$  then  $\models_A (wp(\alpha, \exists n P(n)) \equiv (\exists n)(wp(\alpha, P(n))))$ , where  $n \notin var(\alpha)$ .

Our plan is to precisely define the notion of  $wp(\alpha, Q)$  as being dependent upon the four execution methods of Section 5.3, and then to figure out which of the four resulting  $wp$ 's satisfy P1-P4. We will show that those corresponding to methods D and DT do. However, in the next section we introduce Dijkstra's *guarded commands* (GC) programming language and show that, restricting ourselves to programs in that language, the only notion of  $wp$  which is consistent with the way in which GC is defined in [13] is that corresponding to method D, i.e. depth search with no backtracking. Thus, although there are four independent notions of the weakest precondition of a program, the particular notion of  $wp(\alpha, Q)$  that Dijkstra had in mind in [13] and [14] presupposes the use of method D. Independent work by de Bakker [2], Plotkin (described in [57]) and Hoare [20] (especially the latter) has also indicated that one has to outlaw both infinite computations and "blind alleys" (failures) in order to capture Dijkstra's notion of  $wp$ .

*Definition:* Given a universe  $U$ , a program  $\alpha \in RC$  and a formula  $Q$ , the *weakest precondition* of  $\alpha$  with respect to  $Q$  is defined for methods D, DT, B or BG respectively as follows:

$$\begin{aligned} wp_D(\alpha, J) &= (\langle \alpha \rangle true \wedge [a]Q \wedge \neg loop_a \wedge \neg fail_a), \\ wp_{DT}(\alpha, J) &= (\langle \alpha \rangle true \wedge [a]Q \wedge \neg loop_a), \\ wp_B(\alpha, J) &= (\langle \alpha \rangle true \wedge [a]Q \wedge \neg fail_a), \\ wp_{BG}(\alpha, J) &= (\langle \alpha \rangle true \wedge [a]Q). \end{aligned}$$

Certainly, by definition, given a universe  $U$ , with  $X$  ranging over D, DT, B and BG, a program  $\alpha$  is  $X$ -totally correct wrt  $R$  and  $Q$  iff  $\models_X (R \supset wp_X(\alpha, Q))$ .

Note that all of our four  $wp$ 's satisfy the informal description (\*) in which the word "activation" is now interpreted as "activation using execution method X". In other words, we claim that

It is indeed the case that using method X,  $wp_X(\alpha, Q)$  is the weakest precondition which guarantees that execution of  $\alpha$  using method X will always terminate in a state satisfying Q.

Let us see which of our  $wp$ 's satisfy Dijkstra's properties P1-P4.

**Lemma 5.4:** P1-P3 hold for  $wp_D$ ,  $wp_{DT}$ ,  $wp_B$  and  $wp_{BC}$ .

*Proof:* P1: Since for any  $X \in \{D, DT, B, BC\}$ ,  $(wp_X(\alpha, Q) \supset (\langle \alpha \rangle true \wedge [\alpha]Q))$ ,  $((\langle \alpha \rangle true \wedge [\alpha]Q) \supset \langle \alpha \rangle Q)$ , and  $(\langle \alpha \rangle false = false)$  are all valid, P1 can be seen to follow. We omit the straightforward proofs of P2 and P3. ■

**Lemma 5.5:** There exists an arithmetical universe A, a program  $\alpha \in RC$  and a formula  $P(n)$ , such that P4 does not hold for  $wp_{BC}$  or  $wp_B$ .

*Proof:* Take A to be the universe of pure arithmetic  $N$ , and  $P(n)$  to be  $n \geq x$ . Certainly for any  $n$ , we have  $\models_A (n \geq x \supset (n+1) \geq x)$ . Take  $\alpha$  to be  $(x+0; (x \leftarrow x+1)^*)$ . One can then check that  $\models_N \langle \alpha \rangle true$  and  $\models_N \neg \text{false}$ , both hold, as does  $\models_A [\alpha] \exists n (n \geq x)$ . However  $\models_N \neg [\alpha] (n \geq x)$  does not. ■

**Theorem 5.6:** P4 holds for  $wp_D$  and  $wp_{DT}$ .

*Proof:* Assume  $\models_A \forall n (P(n) \supset P(n+1))$ . Because of  $\text{var}(\alpha)$ , it is immediate that  $(\exists n (wp_D(\alpha, P(n))) \equiv (\langle \alpha \rangle true \wedge \neg \text{loop}_\alpha \wedge \neg \text{fail}_\alpha \wedge \exists n [\alpha] P(n)))$  is A-valid. Also, it is trivial to show that for the same reason, so is  $(\exists n [\alpha] P(n) \supset \exists n \exists m P(n))$ . Assume now that  $\models [\alpha] \exists n P(n)$  holds. We show that  $\models \exists n [\alpha] P(n)$  does too. By  $\models \neg \text{loop}_\alpha$  holding, we know that  $cl(\alpha, J)$  is finite. Consider the set  $J\alpha = \{J, J\alpha\}$ . By virtue of  $\models [\alpha] \exists n P(n)$  holding, there is an integer  $i(J)$  associated with each  $J \in (J\alpha)$ , such that for any  $n$ ,  $J \models P(n)$  whenever  $n \geq i(J)$ . Since  $J\alpha$  is finite (by Lemma 5.2 together with the fact that  $cl(\alpha, J)$  is a finite tree), taking  $m = \max_{J \in (J\alpha)} i(J)$  and observing that for any  $J \in (J\alpha)$  we have  $J \models (P(n) \supset P(m))$  where  $m \geq i$ , we conclude that  $\models \exists n P(n)$  when  $n \geq m$ . ■

For  $w\phi_{DT}$ , it suffices to observe that under the condition  $\neg \text{var}(\alpha)$  we have that  $(\exists n(w\phi_{DT}(\alpha, P(n))) \equiv (\langle \alpha \rangle \text{true} \wedge \neg \text{loop}_\alpha \wedge \exists n[\alpha]P(n)))$  is A-valid. The proof then proceeds exactly as above. ■

Thus, we summarize as follows:

	$w\phi_D$	$w\phi_{DT}$	$w\phi_B$	$w\phi_{BC}$
P1-P3	1	1	1	1
P4	1	1	0	0

and conclude that the properties P1-P4 do *not* give rise to a unique notion of  $w\phi$ ; there are at least two equally plausible definitions which satisfy these properties. We remark that [13] included only P1-P3, and these are satisfied by all the four  $w\phi$ 's. Hence P4, which was added in [14], can be seen to be equivalent to requiring that the program is divergence-free. Wand [68] has essentially shown that nothing weaker than  $w\phi_{DT}$  satisfies P1-P4.

## 5.5 The Guarded Commands Language (GC).

In this section we complete our analysis of the notion of weakest preconditions by restricting ourselves, as did Dijkstra in [13], to a sublanguage of the language RC of regular expressions over assignments and tests, namely to the language of *guarded commands* (GC). We show that only one of the four notions of  $w\phi$ , namely  $w\phi_D$ , is consistent with the manner in which GC was alleged to have been defined in [13]. Since  $w\phi_D$  satisfies P1-P4 of [14] too, we conclude that Dijkstra had been presupposing that method D was to be used in executing the programs in GC.

We define GC as a subset of RC with the same semantics, as follows:

- (1) An assignment  $x \leftarrow e$  is a program in GC.
- (2) For any  $\alpha, \beta \in \text{GC}$  and first-order tests  $P?$  and  $R?$ ,  
 $\alpha; \beta$ ,  
 $(P?; \alpha \cup R?; \beta)$ , and  
 $((P \vee R)?; (P?; \alpha \cup R?; \beta))^*; (\neg P \wedge \neg R)?$  are in GC.

Throughout, we abbreviate the last construct in (2) above, to  $(P?;\alpha * R?;\beta)$ .

One can see that in GC tests do not appear as programs in their own right but only as *guards* preceding "real" statements. Thus, in the alternative construct  $(P?\alpha \cup R?;\beta)$  (written IF  $P \rightarrow \alpha \parallel R \rightarrow \beta$  FI in [13]), either  $\alpha$  or  $\beta$  is executed depending on whether it is P or R which is true. If both are, then one of  $\alpha$  and  $\beta$  is chosen nondeterministically, and if neither is then the statement fails. Thus this construct is a nondeterministic generalization of *if P then  $\alpha$  else  $\beta$* . Similarly, the repetitive construct  $(P?;\alpha * R?;\beta)$  (written DO  $P \rightarrow \alpha \parallel R \rightarrow \beta$  OD in [13]) generalizes *while P do  $\alpha$* .

In [13] the language defined is seemingly somewhat less restrictive. For example,  $(P_1?;\alpha_1 \cup \dots \cup P_n?;\alpha_n)$  is allowed for any  $n > 0$ . However,  $P?;\alpha$ , for all our purposes, is equivalent to  $(P?;\alpha \cup P?;\alpha)$ , and  $(P_1?;\alpha_1 \cup P_2?;\alpha_2 \cup P_3?;\alpha_3)$  to  $(P_1?;\alpha_1 \cup (P_2?;\alpha_2 \cup P_3?;\alpha_3))$ . Also, Dijkstra's *skip* and *abort* statements can be written as  $(true?;x \leftarrow x \cup true?;x \leftarrow x)$  and  $(false?;x \leftarrow x \cup false?;x \leftarrow x)$  respectively; thus GC can be seen to be sufficient. (Remark: *abort* was described in [14] as being a statement that always *fails*, and so is written differently from the statement  $(true?;x \leftarrow x * true?;x \leftarrow x)$  which always diverges and which we call *diverge*.)

In [13] and [14] the semantics of GC was defined using the (informally described) notion of  $wp(\alpha, Q)$ . We rephrase these "definitions" as logical equivalences, noting that a candidate of ours for  $wp$  should satisfy them for any program in GC, in any state. As we shall see, only one of our four  $wp$ 's satisfies them all. The equivalences are:

- D1.  $wp(skip, Q) \equiv Q,$
- D2.  $wp(abort, Q) \equiv false,$
- D3.  $wp(x \leftarrow e, Q) \equiv Q_x^e,$
- D4.  $wp(\alpha; \beta, Q) \equiv wp(\alpha, wp(\beta, Q)),$
- D5.  $wp((P?;\alpha \cup R?;\beta), Q) \equiv ((P \vee R) \wedge (P \supset wp(\alpha, Q)) \wedge (R \supset wp(\beta, Q))),$
- D6.  $wp((P?;\alpha * R?;\beta), Q) \equiv \bigvee_{n=0}^{\infty} (H_n),$   
     where  $H_0 \equiv (\neg P \wedge \neg R \wedge Q),$   
     and  $H_{n+1} \equiv (H_0 \vee wp((P?;\alpha \cup R?;\beta), H_n)).$

*Lemma 5.7:* D1, D2 and D3 hold for  $wp_D$ ,  $wp_{DT}$ ,  $wp_B$  and  $wp_{BC}$ .

*Proof:* D1: For  $skip$ , defined above as  $(true?;x \leftarrow x \cup true?;x \leftarrow x)$  we certainly have  $\models \langle skip \rangle true$ , and similarly, for any  $J \in \Gamma$  one can see that  $\mathcal{R}(skip, J)$  is free of failures and is finite. Also,  $\llbracket skip \rrbracket Q = \llbracket true?;x \leftarrow x \rrbracket Q = \llbracket x \leftarrow x \rrbracket Q = Q_x^x = Q$ . Thus D1 follows.

D2:  $\langle abort \rangle true = (\langle false?;x \leftarrow x \rangle true \vee \langle false?;x \leftarrow x \rangle true) = (false \wedge \langle x \leftarrow x \rangle true) = false$ , and thus since for any  $X \in \{D, DT, B, BC\}$  we have  $wp_X(\alpha, Q) \supseteq \langle \alpha \rangle true$ , we obtain D2.

D3: Since we have  $\models (\langle x \leftarrow e \rangle true \wedge \neg loop_{x \leftarrow e} \wedge \neg fail_{x \leftarrow e})$ , we conclude that for any  $X$  as above,  $wp_X(x \leftarrow e, Q) = \llbracket x \leftarrow e \rrbracket Q = Q_x^e$ . ■

*Theorem 5.8:* For each of  $wp_{DT}$ ,  $wp_B$  and  $wp_{BC}$ , there exist  $\alpha, \beta \in CC$  such that D4 is not valid.

*Proof:* Take  $\alpha$  to be  $(true?;x \leftarrow 1 \cup true?;x \leftarrow 2)$  and  $Q$  to be  $true$ .

DT: Take  $\beta$  to be  $(x=1?;x \leftarrow x \cup x=1?;x \leftarrow x)$ . The left hand side of D4 for this case is  $(\langle \alpha; \beta \rangle true \wedge \neg loop_{\alpha; \beta} \wedge \llbracket \alpha; \beta \rrbracket true)$ . All three conjuncts certainly hold in any state  $J \in N$ . However, the right hand side is  $(\langle \alpha \rangle true \wedge \neg loop_{\alpha} \wedge \llbracket \alpha \rrbracket \langle \beta \rangle true \wedge \llbracket \alpha \rrbracket \neg loop_{\beta} \wedge \llbracket \alpha \rrbracket \llbracket \beta \rrbracket true)$ , and  $\llbracket \alpha \rrbracket \langle \beta \rangle true$  does not hold in any state  $J \in N$ , since for any such  $J$ , we have  $(J, \llbracket 1/x \rrbracket J) \in m(\alpha)$ , but  $\llbracket 1/x \rrbracket J \not\models \langle \beta \rangle true$ .

B: Take  $\beta$  to be  $(x=1?;x \leftarrow x * x=1?;x \leftarrow x)$ . Similarly to the previous case one can see that  $\models_N wp_B(\alpha; \beta, Q)$ , but  $\llbracket \alpha \rrbracket \langle \beta \rangle true$  is not satisfied by any state  $J \in N$  since  $(J, \llbracket 1/x \rrbracket J) \in m(\alpha)$  holds, but  $\llbracket 1/x \rrbracket J \not\models \langle \beta \rangle true$ .

BC: Take  $\beta$  to be any one of the above two. The rest of the reasoning is similar. ■

In order to show that D4 holds for  $wp_D$  we need the following:

*Lemma 5.9:* For any  $\alpha, \beta \in CC$ ,  $\models (fail_{\alpha; \beta} = (fail_{\alpha} \vee \langle \alpha \rangle fail_{\beta}))$ .

(Remark: this lemma should be contrasted with Lemma 5.3(2,3) and the remark following its proof.)

*Proof:* Having Lemma 5.3(2,3) at hand and noting that  $CC = RC$ , all we have left to prove is  $\models (\langle \alpha \rangle fail_{\beta} \supseteq fail_{\alpha; \beta})$  for  $\alpha, \beta \in CC$ . Indeed, the only way there can be a failure in

$cr(\beta, J)$  for some  $J \in (J\alpha)$ , such that that failure disappears in  $cr(\alpha; \beta, J)$ , is in the case where  $cr(\alpha, J)$  has a leaf  $(t, J)$ , the ancestor of which has another descendant which is not a leaf, and furthermore  $cr(\beta, J)$  is simply  $\{(t, J)\}$ . However, one can see that there is no program  $\beta \in CC$  for which  $cr(\beta, J)$  is a singleton. ■

**Theorem 5.10:** For any  $\alpha, \beta \in CC$ , D4 holds for  $w\beta_D$ .

*Proof:* Expanding gives  $w\beta_D(\alpha; \beta, Q) = (\langle \alpha \rangle true \wedge [\alpha; \beta] Q \wedge \neg loop_{\alpha; \beta} \wedge \neg fall_{\alpha; \beta})$ , and similarly  $w\beta_D(\alpha, w\beta_D(\beta, Q)) = (\langle \alpha \rangle true \wedge \neg loop_{\alpha} \wedge \neg fall_{\alpha} \wedge [\alpha] \langle \beta \rangle true \wedge [\alpha] \neg loop_{\beta} \wedge [\alpha] \neg fall_{\beta} \wedge [\alpha] [\beta] Q)$ . By Lemma 5.3(1,2) one direction is seen to follow immediately. Assume now that  $\not\models w\beta_D(\alpha; \beta, Q)$ . Using Lemma 5.3 and Lemma 5.9 for dealing with the clauses involving *loop* and *fall*, we have only to show that  $\not\models [\alpha] \langle \beta \rangle true$  holds. This follows from  $\not\models [\alpha] \neg loop_{\beta}$  and  $\not\models [\alpha] \neg fall_{\beta}$  using Lemma 5.3(4). ■

We now consider D5:

**Lemma 5.11:** For each of  $w\beta_{DT}$ ,  $w\beta_B$  and  $w\beta_{BC}$ , there exists a program  $(P?; \alpha \cup R?; \beta)$  in CC such that D5 is not valid.

*Proof:* Take P, R and Q to be *true?*, and  $\alpha$  to be the program *skip*.

DT: Take  $\beta$  to be *abort*.

B: Take  $\beta$  to be *diverge*.

BC: Take  $\beta$  to be either of the above.

In each case the left hand side of D5 is valid, but the right hand side is not even satisfiable. We omit the details. ■

**Lemma 5.12:** For any  $\alpha, \beta \in CC$ , D5 holds for  $w\beta_D$ .

*Proof:* Straightforward using Lemma 5.3(4) and Lemma 5.9. ■

We now consider D6:

**Theorem 5.13:** For each of  $w\beta_{DT}$ ,  $w\beta_B$  and  $w\beta_{BC}$ , there exists a program  $(P?; \alpha * R?; \beta)$  in CC such that D6 is not valid.

*Proof:* Here too, there is a general structure to the three counter-examples. We present them for each case but omit the tedious, but straightforward, details involved in proving



the claim. In each case, however, one can show that in any state  $J \in N$  such that  $x_J = 0$ , the left hand side of D6 is *true* but the right hand side is not. In fact, the clause  $[P?; \alpha \cup R?; \beta] \langle P?; \alpha \cup R?; \beta \rangle$  *true*, which shows up in  $H_0$  of the right hand side, is the clause which is not true in  $J$ , and which falsifies  $H_1$  for any  $i \geq 2$ .  $H_0$  and  $H_1$  can be checked manually to be false in  $J$ .

Define  $Q$  to be *true*. Taking  $\gamma$  to be the program *abort* for the DT case, *diverge* for the B case, and either of these for the BC case, we define our program  $(P?; \alpha * R?; \beta)$  to be  $((x=0?; x \leftarrow x+3) * (2 \geq x?; x \leftarrow x+1; ((x=1?; x \leftarrow x+1) \cup (x \neq 1?; \gamma))))$ . ■

*Theorem 5.14:* For any  $\alpha, \beta \in CC$ , D6 holds for  $w\rho_D$ .

*Proof:* For simplicity, denote by  $\pi$  the program  $(P?; \alpha \cup R?; \beta)$ , and by  $*\pi$  the program  $(P?; \alpha * R?; \beta)$ . We note that for every  $J$  such that  $J \models w\rho_D(*\pi, Q)$  holds,  $J \models \neg \text{loop}_\pi(*\pi)$  holds, and thus the tree  $ct(*\pi, J)$  is finite. Note that under the same assumption, each leaf of  $ct(*\pi, J)$  is labeled with a state  $\mathcal{J}$  such that  $\mathcal{J} \models (\neg P \wedge \neg R)$ , and also  $\mathcal{J} \models Q$ . We now show that for every  $J \in \Gamma$  such that  $J \models w\rho_D(*\pi, Q)$ , we have  $J \models H_k$ , by induction on  $k$ , where  $k$  is the depth of the tree  $ct(*\pi, J)$ .

If  $k=0$  then  $ct(*\pi, J) = \{(\lambda, F)\}$ , and  $J \models (\neg P \wedge \neg R \wedge Q)$ , so that  $J \models H_0$ . Assume that  $J$  is a state such that  $k$ , the depth of  $ct(*\pi, J)$ , is greater than 0, and assume that  $J \models w\rho_D(*\pi, J)$ . Assume also that for any state  $\mathcal{J}$  such that the depth of  $ct(*\pi, \mathcal{J})$  is  $k'$  and  $k' < k$ , if  $\mathcal{J} \models w\rho_D(*\pi, \mathcal{J})$  holds, then we have  $\mathcal{J} \models H_{k'}$ . We show that  $J \models H_k$  by showing that  $J \models [ \pi ] H_{k-1}$ . This is sufficient because since  $ct(*\pi, J)$  is failure-free and its depth is not 0, it must be the case that  $J \models \langle \pi \rangle$  *true*. Also,  $J \models H_0$ ,  $J \models \text{fall}_\pi$ , and  $J \models \text{loop}_\pi$ .

Take any  $\mathcal{J} \in (J\alpha)$ . Certainly the depth of  $ct(*\pi, \mathcal{J})$  is less than  $k$ . Also, one can show that from the fact that  $J \models w\rho_D(*\pi, Q)$  holds, we can deduce that  $J \models w\rho_D(\pi; *\pi, Q)$  holds too, and then using Lemma 5.12, that  $J \models w\rho_D(*\pi, Q)$  also holds. By the inductive hypothesis we obtain  $\mathcal{J} \models H_{k'}$  for  $k' < k$  (here  $k'$  is the depth of  $ct(*\pi, \mathcal{J})$ ). However, it is easy to establish that for any  $i$ ,  $\models (H_i \supset H_{i+1})$ , so that we also have  $\mathcal{J} \models H_{k-1}$ . Hence  $J \models [ \pi ] H_{k-1}$ . This completes one direction of the lemma.

Conversely, Assume  $J \models H_k$  for some  $k$ . Without loss of generality we can assume that  $J \not\models H_{k'}$  for all  $k' < k$ . If  $k=0$  then trivially  $J \models (\neg P \wedge \neg R \wedge Q)$ , and hence  $J \models w\rho_D(*\pi, J)$ . Assume that  $k > 0$ , and that for any state  $\mathcal{J}$  such that  $\min_i(\mathcal{J} \models H_i)$  is defined and is smaller than  $k$ , we have  $\mathcal{J} \models w\rho_D(*\pi, Q)$ . Certainly by  $J \models H_k$  and  $k > 0$  we

have  $\mathcal{J} \Vdash (\neg P \wedge \neg R \wedge Q)$ , so that  $\mathcal{J} \Vdash (\langle \pi \rangle \text{true} \wedge \neg \text{fail}_{\pi} \wedge \neg \text{loop}_{\pi} \wedge [\pi] H_{k-1})$ . Since  $\mathcal{J} \Vdash \langle \pi \rangle \text{true}$ , we can denote by  $\mathcal{J}$  a state in  $\mathcal{J}\pi$ . We know that  $\mathcal{J} \Vdash [\pi] H_{k-1}$ , and so  $\mathcal{J} \Vdash H_{k-1}$ . Therefore, by the inductive hypothesis we conclude that  $\mathcal{J} \Vdash \text{wp}_D(*\pi, Q)$ , or that  $\mathcal{J} \Vdash (\langle *\pi \rangle \text{true} \wedge \neg \text{fail}_{(*\pi)} \wedge \neg \text{loop}_{(*\pi)} \wedge [*\pi] Q)$ . Now, since  $\mathcal{J} \in (\mathcal{J}\pi)$  and  $\mathcal{J} \Vdash \langle *\pi \rangle \text{true}$ , we have  $\mathcal{J} \Vdash \langle *\pi \rangle \text{true}$ . Similarly we can establish  $\mathcal{J} \Vdash [*\pi] Q$  from  $\mathcal{J} \Vdash [\pi] H_{k-1}$  which implies that  $\mathcal{J} \Vdash [*\pi] Q$  holds for any  $\mathcal{J} \in (\mathcal{J}\pi)$ . Also,  $\mathcal{J} \Vdash \neg \text{fail}_{(*\pi)}$  and  $\mathcal{J} \Vdash \neg \text{loop}_{(*\pi)}$  follow for similar reasons. ■

Thus to summarize, we have the following table, where a 1 indicates validity for all programs in GC:

	$\text{wp}_D$	$\text{wp}_{DT}$	$\text{wp}_B$	$\text{wp}_{RC}$
D1-D3	1	1	1	1
D4-D6	1	0	0	0

We remark that relaxing our restrictions on programs and considering general programs in RC, D4-D6 do not hold in general, even for  $\text{wp}_D$ .

We regard our results in this section as providing rigorous support of the intuition Dijkstra displayed when he designed GC in [14] as a nondeterministic programming language suitable for "total-correctness-oriented" reasoning. Although there is no *a priori* reason for preferring execution method D to any of the others, we have shown that adopting this method in conjunction with the sublanguage GC, results in D1-D6 holding, a fact which nicely gives rise to what Dijkstra calls a "calculus" for computing the weakest precondition of a program, and hence for determining whether a program is totally correct.

## 6. The Mathematics of Diverging and Failing I.

In this chapter we concentrate on some of the mathematical properties of the two concepts of diverging and failing introduced in Chapter 5. Most of the chapter, however, will be concerned with  $loop_\alpha$ . In particular we emphasize the problems of expressing this concept in DL and providing a suitable arithmetical axiomatization of it.

In Section 6.1 we consider the question of obtaining a syntactic equivalent, in DL, of  $loop_\alpha$  and  $fail_\alpha$  for the class of programs RC. In particular, in 6.1.1, we show how a recent theorem of Winklmann [71] serves as the central part in a proof that such an equivalent exists for  $loop_\alpha$ . We then show, in 6.1.2, that an equivalent exists for  $fail_\alpha$  too. Thus, as far as expressive power is concerned,  $loop_\alpha$  and  $fail_\alpha$  add nothing. In Section 6.2 we introduce an extension of DL,  $DL^+$ , in which there is a specially designated primitive for  $loop_\alpha$ . A natural and concise arithmetical axiomatization,  $P^+$ , of  $DL^+$  is given in Section 6.2.2. Section 6.3 is devoted to exhibiting the remarkable similarity in form between the rules for  $\alpha^*$  in  $P$  and  $P^+$ . This observation can be seen to supply a framework to aid when constructing such axiomatizations in general. The framework also supplies a broad perspective for understanding, say, the invariant assertion method of Floyd [17] and Hoare [27] as a special case of arithmetical axiomatizations. Section 6.4 contains an application of these ideas in the form of an arithmetically complete axiomatization of another extension of DL which borrows the  $\text{fix}$  operator of Salwicki [59]. In this extension (ADL) the mechanism introduced for expressing  $loop_\alpha$  is not quite as direct as that of augmenting DL with  $loop_\alpha$  itself (as is essentially done in  $DL^+$ ), but not as indirect as that of adding nothing but rather relying on the equivalent DL-wff of Section 6.1.1.

### 6.1 Diverging and Failing in DL.

It might seem at first that a simple inductive characterization of  $loop_\alpha$  and  $fail_\alpha$  is possible, along the lines, say, of Lemma 5.3(1). There we show that  $loop_{\alpha;\beta}$  is equivalent to  $(loop_\alpha \vee \langle \alpha \rangle loop_\beta)$ . In other words, that being able to determine whether  $\alpha;\beta$  contains a divergence boils down to being able to determine whether  $\alpha$  and  $\beta$

do, given in addition the tools of DL. This task, however, is not quite as simple as it seems. In Sections 6.1.1 and 6.1.2 we focus, respectively, on  $loop_\alpha$  and  $fall_\alpha$ .

### 6.1.1 Expressing $loop_\alpha$ in DL.

**Lemma 6.1:** For every  $\alpha, \beta \in RC$ , assignment  $x \leftarrow e$  and test  $P?$ , the following are valid:

- (1)  $loop_{x \leftarrow e} \equiv false$ ,
- (2)  $loop_{P?} \equiv false$ ,
- (3)  $loop_{\alpha \cup \beta} \equiv (loop_\alpha \vee loop_\beta)$ ,
- (4)  $loop_{\alpha; \beta} \equiv (loop_\alpha \vee \langle \alpha \rangle loop_\beta)$ .

*Proof:* (4) is Lemma 5.3(1). The others follow from the definition of  $cl(\alpha, J)$ . ■

In order to be able to talk about  $\alpha^*$  we allow ourselves, in this chapter, the freedom of writing, say,  $J \models \forall n \langle \alpha^n \rangle P$  instead of "for all  $n$ ,  $J \models \langle \alpha^n \rangle P$  holds". (Recall that  $\alpha^0$  is *true?* and  $\alpha^{n+1}$  is  $\alpha; \alpha^n$ .) We also write  $\exists^\infty n \langle \alpha^n \rangle P$  to read "there exist infinitely many  $n$ 's such that  $\langle \alpha^n \rangle P$  holds".  $\exists^\infty n \langle \alpha^n \rangle P$ , then, asserts that  $\langle \alpha^n \rangle P$  holds of arbitrarily large  $n$ .

**Theorem 6.2:** For every  $\alpha \in RC$ ,  $\models (loop_{\alpha^*} \equiv (\langle \alpha^* \rangle loop_\alpha \vee \forall n \langle \alpha^n \rangle true))$ .

(Remark: In line with the above convention the theorem reads: "In any state  $J$ ,  $J \models loop_{\alpha^*}$  holds iff either  $J \models \langle \alpha^* \rangle loop_\alpha$  holds or for every  $n$  we have  $J \models \langle \alpha^n \rangle true$ .")

*Proof:* As remarked in Chapter 5, by Koening's Lemma for any  $\beta$ ,  $J \models loop_\beta$  holds iff there is an infinite path in  $cl(\beta, J)$ . Now assume  $J \models loop_{\alpha^*}$ . By the construction of  $pcl(\alpha^*, J)$  as  $pcl((true? \cup \alpha; \alpha^*), J)$  it is quite evident that if  $J \models loop_{\alpha^*}$  holds for every  $J \in J(\alpha^*)$  (i.e. if  $J \models [\alpha^*] loop_\alpha$  holds), then an infinite path  $s = (J, J_1, J_2, \dots)$  in  $pcl(\alpha^*, J)$  must be an infinite  $\alpha$ -path, i.e. there must be a subsequence of  $s$  in which every two adjacent states are related via  $m(\alpha)$ . Denote this sequence by  $s' = (J_0, J_1, \dots)$  where  $J_0 = J$  and for every  $n \geq 0$  we have  $(J_n, J_{n+1}) \in m(\alpha)$ . Thus certainly  $(J, J_n) \in m(\alpha^n)$ , and hence  $J \models \langle \alpha^n \rangle true$ .

Conversely, we first note that it is easy to see that  $\models (\langle \alpha^* \rangle loop_\alpha \supset loop_{\alpha^*})$ .

Assume now that  $J \models \forall n \langle \alpha^n \rangle true$ . By the construction of  $pcl(\alpha^*, J)$  this implies that

$pct(\alpha^*, J)$  (and hence also  $ct(\alpha^*, J)$ ) has leaves at arbitrary depth, which by Koenig's lemma implies that  $ct(\alpha^*, J)$  is infinite.

Thus, a divergence in  $\alpha^*$  is due either to a divergence in  $\alpha$  itself after execution of some number of  $\alpha$ 's (*local diverging*), or to being able to run  $\alpha$ 's repeatedly for ever (*global diverging*).

It is immediate then, that the only obstacle to obtaining a straightforward translation of  $loop_\alpha$  into a DL-wff  $Q$  is the fact that  $\forall n \langle \alpha^n \rangle true$  is not a DL-wff. However, we have the following recently established fact:

**Theorem 6.3** (Winklmann [71]): For every  $\alpha \in RC$  and L-wff  $P$  there exists a DL-wff  $Q$  such that  $\vdash(Q = \exists^{\infty} n \langle \alpha^n \rangle P)$ .

The (constructive) proof involves a very subtle argument based on the structure of the set  $J(\alpha^*)$  for some fixed state  $J$ , making a distinction between this set being infinite due to some *repetition* of a state (i.e. some  $J \in J(\alpha^*)$  such that  $J \in J(\alpha; \alpha^*)$ ) and it being infinite but repetition-free. Thus, by noting that  $\forall n \langle \alpha^n \rangle true$  is equivalent to  $\exists^{\infty} n \langle \alpha^n \rangle true$ , we conclude from Lemmas 6.1 and 6.2, and Theorem 6.3:

**Corollary 6.4:** For  $RC$ ,  $loop_\alpha$  is expressible in DL; i.e. for every  $\alpha \in RC$  there exists a DL-wff  $P_\alpha$  such that  $\vdash(P_\alpha = loop_\alpha)$ .

It is easy to generalize the definition of  $ct(\alpha, J)$  to cover the programming languages "array-RC" and "rich-test-RC" which are the sets of programs allowed in array-DL (Section 2.3.1) and rich-test-DL (Section 2.3.3) respectively. These trees are also of finite outdegree and for them too we can define  $J \Vdash loop_\alpha$  to be true iff  $ct(\alpha, J)$  is infinite. We then have

**Theorem 6.5** (Meyer [43]): For every  $\alpha \in \text{array-RC}$  and L-wff  $P$  there exists an array-DL-wff  $Q$  such that  $\vdash(Q = \exists^{\infty} n \langle \alpha^n \rangle P)$ .

**Theorem 6.6** (Winklmann [70]): For every  $\alpha \in \text{rich-test-RC}$  and L-wff  $P$  there exists a rich-test-DL-wff  $Q$  such that  $\vdash(Q = \exists^{\infty} n \langle \alpha^n \rangle P)$ .

**Corollary 6.7:** For array-RC (resp. rich-test-RC),  $loop_\alpha$  is expressible in array-DL (resp. rich-test-DL).

One can define  $ct(\alpha, J)$  for random-DL (Section 2.3.2), a definition which results in trees of infinite outdegree, and then define  $\exists loop_\alpha$  to hold iff  $ct(\alpha, J)$  has an infinite path. Parikh [50] has been able to show that for random-RG,  $loop_\alpha$  is not expressible in random-DL.

Recently Pratt [54] has shown how a plausible definition of  $loop$  for PDL, when the atomic programs in AP (see Chapter 1) are assigned binary relations by the semantics and not some sort of computation trees, gives rise to the possibility of proving that  $loop_\alpha$  is not expressible in PDL.

### 6.1.2 Expressing $fail_\alpha$ in DL.

We now turn to  $fail_\alpha$ . Here too DL is powerful enough to express  $fail_\alpha$  for any  $\alpha \in RG$ . In this case, however, we will need to carry out a very careful analysis of the cases in which a failure node in  $pc(\alpha, J)$  is not deleted when constructing  $ct(\alpha, J)$ . The complication arises in the case of composition (i.e. when  $ct(\alpha; \beta, J)$  has a failure but  $ct(\alpha, J)$  does not). We will see later that for the guarded commands language GC (Section 5.5) this complication vanishes, and in this case the construction of the DL wff  $R_\alpha$  such that  $\models (R_\alpha \equiv fail_\alpha)$  holds is quite straightforward.

Consider now the general set of regular programs RG. We first define inductively the construct  $onenode_\alpha$  such that  $\exists onenode_\alpha$  holds iff  $ct(\alpha, J)$  is a singleton:

$$\begin{aligned} onenode_{x \leftarrow e} &= false, \\ onenode_{P?} &= true, \\ onenode_{\alpha \cup \beta} &= false, \\ onenode_{\alpha; \beta} &= (onenode_\alpha \wedge (onenode_\beta \vee fail_\alpha)), \\ onenode_{\alpha^*} &= false. \end{aligned}$$

Now abbreviate  $(fail_\alpha \wedge \neg onenode_\alpha)$  to  $dfail_\alpha$  (meaning a deep failure of  $\alpha$ ), and  $(fail_\alpha \wedge onenode_\alpha)$  to  $ifail_\alpha$  (immediate failure).

**Lemma 6.8:** For every  $\alpha, \beta \in RG$ , assignment  $x \leftarrow e$  and test  $P?$ , the following are valid:

- (1)  $fail_{x \leftarrow e} = false$ ,
- (2)  $fail_{P?} = \neg P$ ,

$$(3) \text{ fail}_{\alpha \cup \beta} = ((\text{fail}_{\alpha} \wedge \text{fail}_{\beta}) \vee d\text{fail}_{\alpha} \vee d\text{fail}_{\beta}),$$

$$(4) (\text{fail}_{\alpha} \vee \langle \alpha \rangle d\text{fail}_{\beta}) \supset \text{fail}_{\alpha; \beta},$$

$$(5) \text{ fail}_{\alpha; \beta} = \langle \alpha \rangle d\text{fail}_{\beta}$$

*Proof:* We omit the straightforward but rather tedious proofs. ■

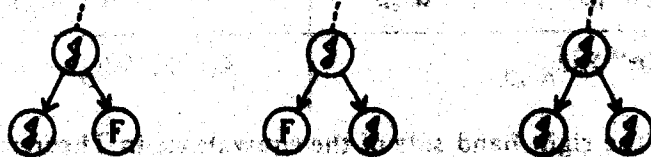
We would like to construct a DL-wff  $\text{other}_{\alpha, \beta}$  such that

$$\text{fail}_{\alpha; \beta} = (\text{fail}_{\alpha} \vee \langle \alpha \rangle d\text{fail}_{\beta} \vee \text{other}_{\alpha, \beta})$$

will be valid. In other words, we would like  $\text{other}_{\alpha, \beta}$  to capture the cases which  $\text{fail}_{\alpha}$  and  $\langle \alpha \rangle d\text{fail}_{\beta}$  do not; i.e. the cases in which there is a failure node in  $ct(\alpha; \beta, J)$  which does not appear in  $ct(\alpha, J)$ , and that furthermore that failure is the result of  $\beta$  being the one-node failure tree in some state  $J \in J_{\alpha}$  (i.e.  $J \neq \text{fail}_{\beta}$ ).

There are precisely three cases in which this situation might occur:

- (1)  $ct(\alpha, J) = \{(\lambda, J)\}$  and  $ct(\beta, J) = \{(\lambda, F)\}$ ,
- (2) in  $ct(\alpha, J)$  there is a node labeled  $J$  which has only one descendant, a leaf labeled  $J$ , such that  $ct(\beta, J) = \{(\lambda, F)\}$ .
- (3) in  $pcct(\alpha, J)$  there is a node labeled  $J$  which has two descendants of which at least one is labeled with some  $J \in \Gamma$ , and such that  $ct(\beta, J) = \{(\lambda, F)\}$ ; i.e. in  $ct(\alpha, J)$ , there is one of:



(Note that in the first two of these diagrams the F-node does not appear in  $ct(\alpha, J)$ , so that in all cases the failure of the form



which occurs when  $\alpha$  is composed with  $\beta$ , cannot be detected by either  $\text{fail}_{\alpha}$  or  $\langle \alpha \rangle d\text{fail}_{\beta}$ .)

For any  $\alpha, \beta \in RC$ , we now supply an inductive construction of the three constructs  $31_{\alpha, \beta}$ ,  $32_{\alpha, \beta}$  and  $33_{\alpha, \beta}$ , corresponding respectively to cases (1)-(3) above. The following theorem can then be seen to follow from this construction

**Theorem 6.9:** For every  $\alpha, \beta \in RC$

$$\vdash (fail_{\alpha; \beta} = (fail_{\alpha} \vee \langle \alpha \rangle fail_{\beta} \vee 31_{\alpha, \beta} \vee 32_{\alpha, \beta} \vee 33_{\alpha, \beta})).$$

Turning to the construction, we note that  $31_{\alpha, \beta}$  is simply  $(onnode_{\alpha} \wedge \neg fail_{\alpha} \wedge \neg fail_{\beta})$ .

We define  $32_{\alpha, \beta}$  and  $33_{\alpha, \beta}$  by induction on  $\alpha$  as follows:

$\alpha$	$32_{\alpha, \beta}$	$33_{\alpha, \beta}$
$x \leftarrow e$	$\langle x \leftarrow e \rangle \neg fail_{\beta}$	false
P?	false	false
$\alpha' U \alpha''$	$(32_{\alpha', \beta} \vee 32_{\alpha'', \beta})$	$(33_{\alpha', \beta} \vee 33_{\alpha'', \beta} \vee (onnode_{\alpha'} \wedge onnode_{\alpha''} \wedge \neg fail_{\beta} \wedge (\neg fail_{\alpha'} \vee \neg fail_{\alpha''})))$
$\alpha'; \alpha''$	$32_{\alpha', (\alpha''; \beta)} \vee \langle \alpha' \rangle 32_{\alpha'', \beta}$	$33_{\alpha', (\alpha''; \beta)} \vee \langle \alpha' \rangle 33_{\alpha'', \beta}$
$\alpha'^*$	$\langle \alpha'^* \rangle 32_{\alpha', \beta}$	$\langle \alpha'^* \rangle (\neg fail_{\alpha'} \wedge \neg fail_{\beta}) \vee \langle \alpha'^* \rangle 33_{\alpha', \beta}$

Note now that the right hand side of the equivalence in Theorem 6.9 is defined using DL-wffs and appearances of  $fail_{\alpha}$  and  $fail_{\beta}$  only. Consequently, Lemma 6.8 and Theorem 6.9 imply:

**Corollary 6.10:** For  $RC$ ,  $fail_{\alpha}$  is expressible in DL; i.e. for every  $\alpha \in RC$  there exists a DL-wff  $R_{\alpha}$  such that  $\vdash (R_{\alpha} = fail_{\alpha})$ .

We remark that for the guarded commands language  $GC$ , we have  $\vdash \neg fail_{\alpha}$  for every  $\alpha \in GC$ . Consequently, it is easy to see that for any  $\alpha \in GC$  we have



$$\models (\neg \exists 1_{\alpha, \beta} \wedge \neg \exists 2_{\alpha, \beta} \wedge \neg \exists 3_{\alpha, \beta}),$$

so that we obtain Lemma 5.9 again, this time as a corollary of Theorem 6.9.

## 6.2 DL Augmented with $loop_{\alpha}$ ( $DL^+$ ).

In this section we introduce an extension of DL,  $DL^+$ , which consists essentially of adding the  $loop_{\alpha}$  construct as a primitive to DL. The virtues of this augmentation are in the ability to reason about divergences directly without having to go through the translation of  $loop_{\alpha}$  into its equivalent DL-wff (Theorem 6.3 and Corollary 6.4). We remark that the DL-wff  $Q$  of Theorem 6.3, and hence  $P_{\alpha}$  of Corollary 6.4 have the unpleasant property of being strongly dependent on the structure of  $\alpha$  and on the variables appearing in  $\alpha$ , so that  $P_{\alpha'}$  cannot be obtained from  $P_{\alpha}$  by substituting  $\alpha'$  for  $\alpha$  throughout. Consequently, proving a formula with an appearance of  $loop_{\alpha}$  will inevitably involve carrying out the transformation of  $loop_{\alpha}$  to  $P_{\alpha}$ , and then reasoning in DL. The point is that the intuition one might have about  $loop_{\alpha}$  is, in a strong sense, lost in the process. On the other hand, the arithmetically complete axiomatization of  $DL^+$  presented in Section 6.2.2 is natural and intuitively appealing.

### 6.2.1 Definitions.

The sets of symbols of  $DL^+$ , the sets of terms and atomic formulae and the set  $RC$  of regular programs, are all as in DL (Section 2.1). The set of  $DL^+$ -wffs is defined as follows:

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

Abbreviations are adopted as in DL, and in addition we abbreviate  $\neg \langle \alpha \rangle^+ \neg P$  to  $[\alpha]^+ P$ , reading "diamond-plus- $\alpha$   $P$ " and "box-plus- $\alpha$   $P$ " respectively.

For the definition of the semantics of  $DL^+$  we adopt the concept of state and universe from Section 2.1, but now we think of the semantics as assigning to every program  $\alpha$  the set of computation trees  $\{\alpha(\alpha, J) \mid J \in I\}$  (see Section 5.2). However, by

virtue of Theorem 5.2 we can continue to refer to  $m(\alpha)$  defined now as  $\{(J, \mathcal{J}) \mid \mathcal{J} \text{ labels a leaf of } cr(\alpha, J)\}$ , while remaining consistent with  $m(\alpha)$  of Chapter 2.

The definition of the set of states satisfying a  $DL^+$ -wff  $P$  is, for atomic formulae and for the clauses  $\neg P$ ,  $(P \vee Q)$ ,  $\exists x P$  and  $\langle \alpha \rangle P$ , taken from Section 2.1. For  $\langle \alpha \rangle^+ P$  we define

$$J \models \langle \alpha \rangle^+ P \quad \text{iff} \quad \text{either } J \models \langle \alpha \rangle P \text{ holds or } cr(\alpha, J) \text{ is infinite.}$$

In other words,  $J \models \langle \alpha \rangle^+ P$  holds iff  $J \models (\langle \alpha \rangle P \vee loop_\alpha)$  does. One can then verify that

$$J \models [\alpha]^+ P \quad \text{iff} \quad \text{both } J \models [\alpha] P \text{ holds and } cr(\alpha, J) \text{ is finite.}$$

From these we obtain our  $DL^+$  versions of  $loop_\alpha$  and  $\neg loop_\alpha$ :

$$\begin{aligned} J \models loop_\alpha & \quad \text{iff} \quad J \models \langle \alpha \rangle^+ \text{false,} \\ J \models \neg loop_\alpha & \quad \text{iff} \quad J \models [\alpha]^+ \text{true.} \end{aligned}$$

With this definition one can see that Corollary 6.4 asserts essentially that  $DL$  and  $DL^+$  are equivalent in expressive power, thus falsifying our conjecture in [25].

We refer the reader to Appendix D in which we exhibit a program with a somewhat nontrivial behavior, the interesting properties of which can be expressed succinctly in  $DL^+$ .

Before proceeding with the axiomatization of  $DL^+$  we would like to exhibit an alternative, but equivalent, definition of the semantics of  $DL^+$ , which justifies the  $^+$ -notation in a rather interesting way, in view of the addition, as in [1] and [56], of an "undefined state" to the grand universe  $\Gamma$ . We mention that this approach was the one taken in our original definition of  $DL^+$  in [25].

Define by  $\Gamma^+$  the set  $\Gamma \cup \{\perp\}$  where  $\perp$  (read "bottom"), the *divergence state*, is a "state" in which, by definition, every  $DL$ -wff is false; i.e.  $\{\perp \models P\} = \emptyset$ . Note then, that  $\perp \not\models P$  and  $\perp \not\models \neg P$  both hold, so that  $\perp \models P$  and  $\perp \not\models \neg P$  are not the same.

Now let  $m^+(\alpha) = (m(\alpha) \cup \{(J, \perp) \mid J \models loop_\alpha\})$ , and (solely for the sake of this definition) let  $J \models \mathcal{J}$  stand for  $(J, \mathcal{J}) \in m^+(\alpha)$ . If we now write

$$\mathcal{J} \vdash \langle \alpha \rangle P \text{ iff } \exists \mathcal{J} (\mathcal{J} \alpha \mathcal{J} \wedge \mathcal{J} \vdash P)$$

then  $[\alpha]P$  defined as  $\neg \langle \alpha \rangle \neg P$  should read

$$\mathcal{J} \vdash [\alpha]P \text{ iff } \forall \mathcal{J} (\mathcal{J} \alpha \mathcal{J} \supset \mathcal{J} \vdash P)$$

rather than with  $\mathcal{J} \vdash P$  on the right. On the other hand one can see that  $\forall \mathcal{J} (\mathcal{J} \alpha \mathcal{J} \supset \mathcal{J} \vdash P)$  asserts that  $\mathcal{J} \vdash [\alpha]P$  and that furthermore  $(\mathcal{J}, \perp) \notin m^*(\alpha)$  (otherwise we would have had  $\mathcal{J} \alpha \perp$  and  $\perp \vdash P$ ). And so  $\forall \mathcal{J} (\mathcal{J} \alpha \mathcal{J} \supset \mathcal{J} \vdash P)$  asserts the same as  $[\alpha]P$  above. Thus we can define

$$\begin{aligned} \mathcal{J} \vdash [\alpha]^+ P & \text{ iff } \forall \mathcal{J} (\mathcal{J} \alpha \mathcal{J} \supset \mathcal{J} \vdash P), \\ \mathcal{J} \vdash \langle \alpha \rangle P & \text{ iff } \exists \mathcal{J} (\mathcal{J} \alpha \mathcal{J} \wedge \mathcal{J} \vdash P), \\ \mathcal{J} \vdash [\alpha]P & \text{ iff } \forall \mathcal{J} (\mathcal{J} \alpha \mathcal{J} \supset \mathcal{J} \vdash \neg P), \\ \mathcal{J} \vdash \langle \alpha \rangle^+ P & \text{ iff } \exists \mathcal{J} (\mathcal{J} \alpha \mathcal{J} \wedge \mathcal{J} \vdash \neg P). \end{aligned}$$

However, in the sequel we abolish the artificial state  $\perp$  and treat  $\langle \alpha \rangle^+ P$  as the abbreviation of  $(\langle \alpha \rangle P \vee \text{loop}_\alpha)$  which we used above.

### 6.2.2 Axiomatization of $DL^+$ .

Let us first gather some of the properties of  $\langle \alpha \rangle^+$  and  $[\alpha]^+$ , most of which have been proved previously for  $\text{loop}_\alpha$ :

*Lemma 6.11:* For every  $\alpha, \beta \in RC$ , assignment  $x \leftarrow e$ , test  $Q?$  and  $DL^+$ -wffs  $P$  and  $R$ , the following are valid

- (1)  $[\alpha]^+ P \equiv ([\alpha]P \wedge [\alpha]^+ \text{true})$ ,
- (2)  $\langle \alpha \rangle^+ P \equiv (\langle \alpha \rangle P \vee \langle \alpha \rangle^+ \text{false})$ ,
- (3)  $[x \leftarrow E]^+ \text{true}$ ,
- (4)  $[Q?]^+ \text{true}$ ,
- (5)  $[\alpha; \beta]^+ P \equiv [\alpha]^+ [\beta]^+ P$ ,
- (6)  $\langle \alpha; \beta \rangle^+ P \equiv \langle \alpha \rangle^+ \langle \beta \rangle^+ P$ ,
- (7)  $[\alpha \cup \beta]^+ P \equiv ([\alpha]^+ P \wedge [\beta]^+ P)$ ,
- (8)  $\langle \alpha \cup \beta \rangle^+ P \equiv (\langle \alpha \rangle^+ P \vee \langle \beta \rangle^+ P)$ ,
- (9)  $[\alpha]^+ (P \wedge R) \equiv ([\alpha]^+ P \wedge [\alpha]^+ R)$ ,
- (10)  $\langle \alpha \rangle^+ (P \vee R) \equiv (\langle \alpha \rangle^+ P \vee \langle \alpha \rangle^+ R)$ .

*Proof:* We prove (5).  $[\alpha; \beta]^+P$  is, by definition,  $([\alpha; \beta]P \wedge [\alpha; \beta]^+true)$  or  $([\alpha][\beta]P \wedge [\alpha; \beta]^+true)$ . However, since by Lemma 5.3(1) we have  $[\alpha; \beta]^+true = ([\alpha]^+true \wedge [\alpha][\beta]^+true)$ , we conclude that  $[\alpha; \beta]^+P = ([\alpha]^+true \wedge [\alpha][\beta]P \wedge [\beta]^+true) = ([\alpha]^+true \wedge [\alpha][\beta]^+P) = [\alpha]^+[\beta]^+P$ . We omit the proofs of other parts. ■

Note how the choice of  $\langle \alpha \rangle^+P$  to abbreviate the disjunction of  $\langle \alpha \rangle P$  and  $loop_\alpha$  is paying off in elegance, as in e.g. Lemma 6.11(5,6). Also, we now use our convention of allowing  $\alpha^n$  to appear in our discussions, in order to show how Theorem 6.2 gives rise to an extremely concise characterization of  $loop_\alpha$ .

**Theorem 6.12:** For every  $\alpha \in RC$ ,

- (1)  $\vdash \langle \alpha^* \rangle^+false = \forall n \langle \alpha^n \rangle^+true$ ,
- (2)  $\vdash [\alpha^*]^+true = \exists n \langle \alpha^n \rangle^+false$ .

*Proof:* By Theorem 6.2  $\langle \alpha^* \rangle^+false$  is equivalent to  $\langle \alpha^* \rangle \langle \alpha \rangle^+false \vee \forall n \langle \alpha^n \rangle^+true$  which can be seen to be equivalent to  $(\exists n \langle \alpha^n \rangle^+false \vee \forall n \langle \alpha^n \rangle^+true)$  or  $(\exists loop_\alpha n \vee \forall n \langle \alpha^n \rangle^+true)$ .

*Claim:*  $\vdash (\exists loop_\alpha n \supset \forall n (loop_\alpha n \vee \langle \alpha^n \rangle^+true))$ .

*Proof:* Assume  $\neg \exists loop_\alpha n$ . Let  $n_0 = \min\{n \mid \neg \exists loop_\alpha n\}$ . Surely then, for every  $n < n_0$  we have  $\exists loop_\alpha n$ , for otherwise  $\exists loop_\alpha n$  would have to hold for some such  $n$ . Also, for any  $n \geq n_0$  we have  $\neg \exists loop_\alpha n$ , and thus the claim is proved.

With this claim established it is easy to see that  $(\exists loop_\alpha n \vee \forall n \langle \alpha^n \rangle^+true)$  is equivalent to  $\forall n (loop_\alpha n \vee \langle \alpha^n \rangle^+true)$  or  $\forall n \langle \alpha^n \rangle^+true$ . (2) follows from (1) by definition of  $[\alpha^*]^+$ . ■

Now let  $A$  be any arithmetical universe, and consider the axiom system  $P^+$  for  $DL^+$ , defined as  $P$  of Section 3.2 augmented with the following axioms and rules:

*Axioms:*

- (O)  $[\alpha]^+P = ([\alpha]P \wedge [\alpha]^+true)$ ,
- (P)  $[x \leftarrow E]^+true$ ,
- (Q)  $[Q?]^+true$ ,
- (R)  $[\alpha; \beta]^+true = [\alpha]^+[\beta]^+true$ ,
- (S)  $[\alpha \cup \beta]^+true = ([\alpha]^+true \wedge [\beta]^+true)$ ,

*Inference rules:*

(T)  $P(n+1) \supset [\alpha]^+ P(n), \neg P(0)$

---

$P(n) \supset [\alpha^*]^+ true$

for an L-wff P with free n,  
s.t.  $nf\ var(\alpha)$ ,

(U)  $P \supset \langle \alpha \rangle^+ P$

---

$P \supset \langle \alpha^* \rangle^+ false$

Provability in  $P^+$  is as defined in Section 3.2. Here too we first establish the soundness of  $P^+$  by showing the soundness of rules (T) and (U):

*Lemma 6.13:* For any L-wff  $P(n)$  and  $\alpha \in RC$ , where  $nf\ var(\alpha)$ ,  
if  $\vDash_A (P(n+1) \supset [\alpha]^+ P(n))$  and  $\vDash_A \neg P(0)$ , then  $\vDash_A (P(n) \supset [\alpha^*]^+ true)$ .

*Proof:* Assume the two hypotheses, and also assume that  $J \vDash P(n)$  holds. Without causing confusion we can denote  $n_j$  by  $n$ . We have to show that  $ct(\alpha^*, J)$  is finite. It is easy to see that a chain  $J_0, J_1, J_2, \dots$  such that  $J_0 = J$  and  $\forall i (J_i \supset J_{i+1})$  is impossible, for by the first hypothesis it would imply  $J_n \vDash P(0)$ , contradicting the second. Similarly, by the first assumption, for any  $J \in J(\alpha^*)$  we know that  $ct(\alpha, J)$  is finite, and hence by Theorem 6.2 there is no way for  $\alpha^*$  to diverge. ■

*Lemma 6.14:* For any universe  $U$ , DL<sup>+</sup>-wff  $P$  and  $\alpha \in RC$ , if  $\vDash_U (P \supset \langle \alpha \rangle^+ P)$   
then  $\vDash_U (P \supset \langle \alpha^* \rangle^+ false)$ .

*Proof:* Assume  $\vDash_U (P \supset \langle \alpha \rangle^+ P)$ , and  $J \vDash P$ . If  $J \vDash \langle \alpha^* \rangle loop_\alpha$  holds, then by Theorem 6.2 so does  $J \vDash loop_{\alpha^*}$ , or  $J \vDash \langle \alpha^* \rangle^+ false$ . Assume then, that  $J \vDash \langle \alpha^* \rangle loop_\alpha$ . We show that  $\forall n \langle \alpha^n \rangle true$ . Indeed, by  $\vDash_U (P \supset \langle \alpha \rangle^+ P)$  and  $J \vDash P$  we can show, by induction on  $n$ , that for all  $n$  we have  $J \vDash \langle \alpha^n \rangle true$ . ■

As in  $P$ , we remark that rule (U) can be replaced by the (valid) induction axiom scheme

$$[a^*](P \supset \langle a \rangle^+ P) \supset (P \supset \langle a^* \rangle^+ \text{false})$$

which is derivable from  $P^+$ , and from which (using parts of  $P^+$  rule (U)) can be derived.

Thus, from Theorem 3.6 and Lemmas 6.11, 6.13 and 6.14 we obtain:

**Theorem 6.15 (A-soundness of  $P^+$ ):** For any  $DL^+$ -wff  $P$ , if  $\vdash_{P^+} P$  then  $\vDash_A P$ .

Here too we would like to apply the Theorem of Completeness (Theorem 3.1) to obtain an arithmetical completeness result for  $P^+$ . One can state a slightly more general version of that theorem, in which more than one functional symbol is added to  $L$ . We omit the precise statement of such a theorem, but note that the proof of it is a trivial rephrasing of the proof of Theorem 3.1, and that adding  $N^+$  and  $N^*$  defined respectively as  $\langle a \rangle^+$  and  $\langle a \rangle^*$ , results in  $DL^+$  being the  $(N^+, N^*)$ -extension of the first-order language  $L$ . Thus, here too, having already established Theorems 3.3 and 3.11 for  $\langle a \rangle^+$  and  $\langle a \rangle^*$ , we need basic completeness results for  $\langle a \rangle^+$  and  $\langle a \rangle^*$  too. Before that, though, we must have

**Theorem 6.16:**  $L$  is  $A$ -expressive for  $DL^+$ .

*Proof:* Trivial using Corollary 6.4 and Theorem 3.21. ■

We now have:

**Lemma 6.17:** The following are derived rules of  $P^+$ :

$$(H^*) \frac{P \supset Q}{\langle a \rangle^+ P \supset \langle a \rangle^+ Q}$$

$$\langle a \rangle^+ P \supset \langle a \rangle^+ Q$$

$$(T) \frac{R \supset \exists n P(n) \quad , \quad P(n+1) \supset [a]^+ P(n) \quad , \quad \neg P(0)}{R \supset [a]^+ \text{true}}$$

for an  $L$ -wff  $P$  with free  $n$ ,

$$R \supset [a]^+ \text{true}$$

is. of. var( $a$ ),

$$(U) \frac{R \supset P \quad , \quad P \supset \langle a \rangle^+ P}{R \supset \langle a^* \rangle^+ \text{false}}$$

$$R \supset \langle a^* \rangle^+ \text{false}$$

*Proof:* Trivial. ■

We will now combine the two phases (treated separately for DL in Section 3.2) of (a) showing how to A-validate the premises of (T') and (U') when their conclusions are A-valid, and (b) showing box<sup>+</sup>- and diamond<sup>+</sup>-completeness:

**Theorem 6.18 (Box<sup>+</sup>-completeness Theorem):** For every  $\alpha \in RC$  and L-wffs R and Q, if  $\vDash_A (R \supset [\alpha]^+ Q)$  then  $\vdash_{P^+} (R \supset [\alpha]^+ Q)$ .

*Proof:* Since  $\vDash_A (R \supset [\alpha]^+ Q)$ , prove  $R \supset [\alpha]^+ Q$  in  $P$  by Theorem 3.9, then prove  $R \supset [\alpha]^+ true$  in  $P^+$  as follows, and use axiom (O) to combine the results. The existence of a proof, in  $P^+$ , of  $R \supset [\alpha]^+ true$  is established by induction on the structure of  $\alpha$ , with the only non-trivial case being  $\alpha^*$ . For this case, if  $\vDash_A (R \supset [\alpha^*]^+ true)$  holds, then apply the derived rule (T') with  $P(n)$  taken simply as an arithmetical equivalent of  $[\alpha^n]^+ false$ . By Theorem 6.12 we have  $\vDash (\llbracket \alpha^* \rrbracket^+ true \equiv \exists n P(n))$ , and so the premises of (T') can be seen to hold. ■

**Theorem 6.19 (Diamond<sup>+</sup>-completeness Theorem):** For every  $\alpha \in RC$  and L-wffs R and Q, if  $\vDash_A (R \supset \langle \alpha \rangle^+ Q)$  then  $\vdash_P (R \supset \langle \alpha \rangle^+ Q)$ .

*Proof:* As in the previous theorem. Here for  $R \supset \langle \alpha^* \rangle^+ false$  the derived rule (U') is applied, taking P to be an arithmetical equivalent of  $\langle \alpha^* \rangle^+ false$  itself. One can show that  $\vDash (loop_{\alpha^*} \supset (\langle \alpha \rangle loop_{\alpha^*} \vee loop_{\alpha^*}))$ , which establishes the A-validity of the premise  $P \supset \langle \alpha \rangle^+ P$ . (In fact, the implication for this case can be replaced by an equivalence.) ■

As we remarked at the end of Section 3.2 for the Box-completeness Theorem, here too we can satisfy the premises of (U') by a "strongest  $\langle \rangle^+$ -consequent" giving rise to an alternative proof of Theorem 6.19; take P to be an arithmetical equivalent of  $(\langle (\alpha^-)^* \rangle R \wedge \langle \alpha^* \rangle^+ false)$ . Trivially if  $\vDash_A (R \supset \langle \alpha^* \rangle^+ false)$  then  $\vDash_A (R \supset P)$ , and we leave to the reader to show the more subtle fact that  $\vDash_A (P \supset \langle \alpha \rangle^+ P)$  holds too. In the next section we concentrate on the rules for  $\alpha^*$  in  $P$  and  $P^+$ , and on the way in which we were able to A-validate their premises in order to obtain the basic completeness results.

We conclude:

**Theorem 6.20 (Arithmetical Soundness and Completeness for DL<sup>+</sup>):** For any DL<sup>+</sup>-wff P,  $\vDash_A P$  iff  $\vdash_{P^+} P$ .

*Proof:* One direction is Theorem 6.15, and the other follows from Theorems 3.1, 6.16, 6.18 and 6.19, and the derived rule (H<sup>\*</sup>) in Lemma 6.17. ■

Appendix D contains an example of an interestingly behaving program and a proof in  $P^*$  of some of its properties expressed in  $DL^*$ .

### 6.3 A Pattern of Reasoning.

We now exhibit a rather surprising similarity in the way in which the rules for  $\alpha^*$  in  $P$  and  $P^*$  have been developed, and the part they play in proving the arithmetical completeness of these systems. We will see that there are four concepts involved,  $\{ \alpha^* \} Q$ ,  $\langle \alpha^* \rangle Q$ ,  $\{ \alpha^* \}^+ true$  and  $\langle \alpha^* \rangle^+ false$ , two of which are of universal and two of existential nature. For each, a "descending" induction rule involving  $P(n)$  can be constructed directly from knowing e.g. that  $\{ \alpha^* \} Q$  is  $\forall n \{ \alpha^* \}^n Q$  and that  $\{ \alpha^* \}^+ true$  is  $\exists n \{ \alpha^* \}^n false$ .

Furthermore, these rules are sufficient to make an appropriate axiom system arithmetically complete because it is straightforward to find a  $P(n)$  which  $A$ -validates the premises of each when its conclusion is  $A$ -valid (i.e.  $P(n)$  is the appropriate "weakest antecedent"). For the two "universal" concepts, there exist similar "ascending" rules which are dual to the descending ones. These are sufficient in the above sense too, this time the  $P(n)$  is a "strongest consequent". For the cases where two rules exist, they can be collapsed into one rule free of occurrences of  $n$ ; i.e. not confined to arithmetical universals. The premises of this rule are  $A$ -validated by (what amounts to) both the weakest antecedent and the strongest consequent. Finally, since the resulting rules are the derived rules (I'), (J'), (T') and (U'), we observe that from them the rules (I), (J), (T) and (U) can be obtained.

In the sequel, for brevity, we use  $P$  to denote  $P(n)$ ,  $P'$  to denote  $P(n+1)$  and  $P^0$  to denote  $P(0)$ . We present these observations by working on all four concepts analogously:



The concepts involved are

$[\alpha^*]Q$	$\langle \alpha^* \rangle Q$
$[\alpha^*]^+ true$	$\langle \alpha^* \rangle^+ false.$

The concise arithmetical characterizations of these concepts are

$\forall n[\alpha^n]Q$	$\exists n\langle \alpha^n \rangle Q$
$\exists n[\alpha^n]^+ false$	$\forall n\langle \alpha^n \rangle^+ true.$

An ascending inductive rule of inference can now be constructed by introducing P, having R imply  $QnP$  where the quantifier Q is determined by the arithmetical characterization, and having  $P^0$  imply the rightmost subformula in that characterization:

$\frac{R \supset \forall nP, P^0 \supset [\alpha]P, P^0 \supset Q}{R \supset \forall n[\alpha^n]Q}$	$\frac{R \supset \exists nP, P^0 \supset \langle \alpha \rangle P, P^0 \supset Q}{R \supset \exists n\langle \alpha^n \rangle Q}$
$\frac{R \supset \exists nP, P^0 \supset [\alpha]^+ P, P^0 \supset false}{R \supset \exists n[\alpha^n]^+ false}$	$\frac{R \supset \forall nP, P^0 \supset \langle \alpha \rangle^+ P, P^0 \supset true}{R \supset \forall n\langle \alpha^n \rangle^+ true}$

The premises of these rules are A-validated (when the consequents are A-valid) by taking P to be A-equivalent to

$[\alpha^n]Q$	$\langle \alpha^n \rangle Q$
$[\alpha^n]^+ false$	$\langle \alpha^n \rangle^+ true.$

We could have stopped here; the above rules are sound and "complete", and will enable a completeness theorem to go through. We continue however, as explained above.

Since we have the duality principle (see [52])  $\vdash((R \supset [\beta]Q) \equiv (\langle \beta^- \rangle R \supset Q))$ , natural "ascending" rules, which are constructed dually to the descending ones are

$R \supset P^0, P \supset [\alpha]P^*, \exists n P \supset Q$ ----- $R \supset \forall n [\alpha^n]Q$	<i>no rule</i>
<i>no rule</i>	$R \supset P^0, P \supset \langle \alpha \rangle^+ P^*, \exists n P \supset \text{true}$ ----- $R \supset \forall n \langle \alpha^n \rangle^+ \text{true}$

Recalling  $m(\alpha^-) = \{[1, 1] \} \{[\alpha]\}$ , the premises of these rules can be seen to be A-validated by

$\langle (\alpha^-)^n \rangle R$	<i>no rule</i>
<i>no rule</i>	$\langle (\alpha^-)^n \rangle R \wedge \langle \alpha^n \rangle^+ \text{false.}$

The ascending and descending rules can be collapsed (by virtue of e.g. the fact that  $(\forall n [\alpha^n]P \supset [\alpha] \forall n [\alpha^n]P)$ , or  $([\alpha^*]P \supset [\alpha] [\alpha^*]P)$ ), giving the unified rules

$R \supset P, P \supset [\alpha]P, P \supset Q$ ----- $R \supset [\alpha^*]Q$	<i>no rule</i>
<i>no rule</i>	$R \supset P, P \supset \langle \alpha \rangle^+ P, P \supset \text{true}$ ----- $R \supset \langle \alpha^* \rangle^+ \text{false}$

The premises of these rules are A-validated by both

$[\alpha^*]Q$ and $\langle (\alpha^-)^* \rangle R$	<i>no rule</i>
<i>no rule</i>	$\langle \alpha^* \rangle^+ \text{false}$ and $\langle (\alpha^-)^* \rangle R \wedge \langle \alpha^* \rangle^+ \text{false.}$

Now the "arms and legs" of these rules can be pruned, noting e.g., that from having proved  $R \supset \exists n P, P^0 \supset Q$  and  $\forall n (P \supset \langle \alpha^n \rangle^+ P^0)$ , we can deduce  $R \supset \langle \alpha^* \rangle^+ Q$  using validities of first order logic (included as axioms in (A)). Thus we obtain the final rules

$\frac{P \supset [\alpha]P}{P \supset [\alpha^*]P}$	$\frac{P' \supset \langle \alpha \rangle P}{P' \supset \langle \alpha^* \rangle P^0}$
$\frac{P' \supset [\alpha]^+ P, \neg P^0}{P \supset [\alpha^*]^+ true}$	$\frac{P' \supset \langle \alpha \rangle^+ P}{P \supset \langle \alpha^* \rangle^+ false}$

The name given to the constructs used to A-validate the premises (i.e. the L-wff  $P(n)$  which one needs to "invent" in order to be able to carry out a proof) is:

<i>invariant</i>	<i>convergent</i>
??	<i>divergent</i> (we suggest).

We would appreciate suggestions on suitable names for the "??".

We would like the reader to consider the virtues of conducting this reasoning for the language of regular expressions over assignments and tests. Consider how much more obscure the observations of this section would have been if we were to reason about, say, the *while* statement, instead of about  $\alpha^*$ . In our opinion  $\alpha^*$  captures the raw essence of *iterating* in programming languages, just as  $\alpha \cup \beta$  captures the essence of *branching* and  $\alpha; \beta$  the essence of *sequencing*. For the programming language designer who is interested in a deterministic language or in a more "disciplined" nondeterministic one, we can recommend means of restricting the generality of these constructs (e.g. *if-then-else* and *while*, or simply Dijkstra's [14] guarded commands language GC (Section 5.5)). Note how the *invariant assertion* method of Floyd [17], as described by Hoare's *while* rule [27] (see Section 3.3), has been shown to fall out of this general pattern of arithmetically complete rules as a special case.

#### 6.4 DL with an Iteration Quantifier (ADL).

In this section we consider a different extension of DL, in which, instead of  $loop_\alpha$ , a primitive expressing  $\forall n \langle \alpha^n \rangle P$  is added. Note then, that it is immediate that by

Lemma 6.1 and Theorem 6.2,  $loop_\alpha$  can be expressed, and there is no need to construct Winklmann's  $Q$  of Theorem 6.3. We will supply an arithmetically complete axiomatization of ADL, stressing the fact that the rules were constructed quite easily guided by the observations of Section 6.3 concerning rules for  $\alpha^*$ .

Formally, ADL is defined similarly to  $DL^+$  in Section 6.1.1; i.e. terms, atomic formulae, states, universes etc. are the same. The set of ADL-wffs is defined as

- (1) Any atomic formula is an ADL-wff,
- (2) For any ADL-wffs  $P$  and  $Q$ ,  $\alpha$  in  $RC$  and variable  $x$ ,  
 $\neg P$ ,  $(P \vee Q)$ ,  $\exists x P$ ,  $\alpha \rightarrow P$  and  $(\bigcap \alpha)P$  are ADL-wffs.

For defining the semantics all we need is the following: For any state  $\mathcal{J} \in \Sigma$ , ADL-wff  $P$  and  $\alpha \in RC$ , define  $\mathcal{J} \models (\bigcap \alpha)P$  to hold iff  $\mathcal{J} \models \alpha^n P$  holds for every  $n \geq 0$ . Using our convention then, we have

$$\mathcal{J} \models (\bigcap \alpha)P \iff \forall n (\alpha^n \models P).$$

The construct  $(\bigcap \alpha)P$  is the iteration quantifier used by Subwilo [55] and his colleagues in their work on algorithmic logic (see e.g. [7] for a survey of this work). Certainly we have, following Theorem 6.2:

**Lemma 6.21:** For every  $\alpha \in RC$ , there exists an ADL-wff  $P_\alpha$  such that  $\mathcal{J} \models loop_\alpha \iff \mathcal{J} \models P_\alpha$ .

Thus, ADL is at least as expressive as  $DL^+$ . We remark that it is not clear whether the  $\exists^*$  in Theorems 6.3, 6.5 and 6.6 can be replaced by  $\forall$ . If it can, then DL,  $DL^+$  and ADL would all be equal in expressive power for the regular, array and rich-text versions of DL. At the moment, though, we cannot prove that  $DL^+$  and ADL are equal for any of these cases. However, for our purposes the following is sufficient, and can be shown easily using  $ITR_0$  of Theorem 3.2:

**Theorem 6.22:** L is  $\Lambda$ -expressive for ADL.

Here too we will obtain our completeness result by applying Theorem 3.1. However, we omit most of the proof, noting that it follows in the lines of Theorems 3.12, 4.15 and 6.20.

**Lemma 6.23:** For any program  $\alpha$  and L-wffs  $R$  and  $Q$ ,  
if  $\vDash_A (R \supset Q)$  then  $\vDash_A ((\bigcap \alpha)R \supset (\bigcap \alpha)Q)$ .

*Proof.* We omit this slightly tedious but nevertheless straightforward proof. ■

Having Lemma 6.23 at hand, we add the following rule to  $\mathcal{P}$ :

$$(R) \quad \frac{P \supset Q}{(\bigcap \alpha)P \supset (\bigcap \alpha)Q}$$

We also add the rules:

$$(S) \quad \frac{R \supset P(n) \ , \ P(n+1) \supset \langle \alpha \rangle P(n) \ , \ P(0) \supset Q}{R \supset (\bigcap \alpha)Q} \quad \begin{array}{l} \text{For an L-wff } P \text{ with free } n, \\ \text{s.t. } n \notin \text{var}(\alpha), \end{array}$$

$$(T) \quad \frac{P(n+1) \supset [\alpha]P(n)}{P(n) \supset \neg(\bigcap \alpha)\neg P(0)} \quad \begin{array}{l} \text{For an L-wff } P \text{ with free } n, \\ \text{s.t. } n \notin \text{var}(\alpha), \end{array}$$

(S) and (T) are obtained from the following rules, which in turn follow quite effortlessly from considerations similar to those described in Section 6.3:

$$\frac{R \supset \forall n P \ , \ P' \supset \langle \alpha \rangle P \ , \ P^0 \supset Q}{R \supset \forall n \langle \alpha^n \rangle Q} \quad \text{and} \quad \frac{R \supset \exists n P \ , \ P' \supset [\alpha]P \ , \ P^0 \supset Q}{R \supset \exists n [\alpha^n]Q}$$

We do not know of a duality principle, or of any other way for doing away with the indices in rule (S). Denoting the resulting axiom system by  $\mathcal{P}(\cap)$ , we have

**Theorem 6.24 (Arithmetical Soundness and Completeness for ADL):** For any ADL-wff  $P$ ,  
 $\vDash_A P$  iff  $\vdash_{\mathcal{P}(\cap)} P$ .

*Proof.* Apply Theorem 3.1, and in the appropriate place (i.e. when proving that whenever  $\vDash_A (R \supset (\bigcap \alpha)P)$  holds then so does  $\vdash_{\mathcal{P}(\cap)} (R \supset (\bigcap \alpha)P)$ ) use the above two derived rules, showing that their premises can be made A-valid when their conclusions are, by taking  $P(n)$  to be arithmetical equivalents of  $\langle \alpha^n \rangle Q$  and  $[\alpha^n]Q$  respectively. ■

## 7. The Mathematics of Diverging and Failing II.

In this chapter we consider generalizing the methods developed in Chapter 6 in order to facilitate reasoning about the divergence and failure of programs other than those in the set  $RC$ ; e.g. those in the set  $CF$  of recursive programs defined in Chapter 4.

In Section 7.1 we extend the definition of computation trees to  $CF$ , thus giving rise to  $loop_\alpha$  and  $fail_\alpha$  defined over this set. In Section 7.2 we consider the problem of whether, for  $\alpha \in CF$ ,  $loop_\alpha$  and  $fail_\alpha$  can be expressed as CFDL-wffs; in particular we provide the analogue, for  $CF$ , of Theorem 6.2 and state as an open problem that of obtaining the analogue of Theorem 6.3. Section 7.3 is concerned with augmenting CFDL with  $loop_\alpha$  for  $\alpha \in CF$ , giving  $CFDL^*$ , the resulting axiomatization (Section 7.3.2) not being quite as elegant as that of  $DL^*$  in Section 6.2.2. The results of these sections raise the question of whether there is an inherent difficulty in reasoning about recursive programs.

Section 7.4 is devoted to describing a general way in which notions of diverging and failing can be defined on the basis of sets of strings over the alphabet of assignments and tests in the spirit of the r.e. programs of Section 2.3.5. We denote these concepts by *lang-loop* (language loop) and *lang-fail* respectively, and use them to clarify the intuition with which the seemingly ad hoc computation trees were constructed for  $RC$  and  $CF$  in Sections 5.2 and 7.1.

### 7.1 Computation Trees for Recursive Programs.

The preliminary computation tree  $pcr(\alpha, J)$  of a program  $\alpha \in CF$  and a state  $J \in \Gamma$  is defined by induction on the structure of  $\alpha$  precisely as in Section 5.2, but with the clause

$$(5) \quad pcr(\tau^*(f), J) = pcr(\text{false?} \cup \tau(\tau^*(f)), J)$$

replacing the  $\alpha^*$  clause. Informally, to construct  $pcr(\tau^*(f), J)$  one starts constructing  $pcr(\text{false?} \cup \tau(\tau^*(f)))$ , and whenever, so to speak, " $pcr(\tau^*(f))$ " has to be attached to a node labeled  $J$ ,  $pcr(\tau^*(f), J)$  is attached instead. Of course, as was the case with  $pcr(\alpha^*, J)$ ,

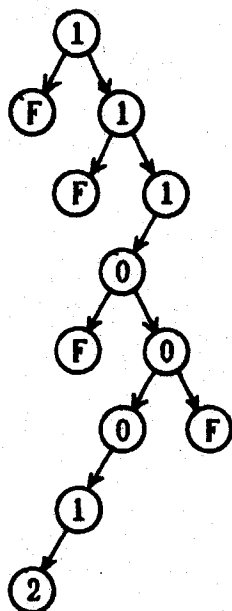
this process can lead to an infinite tree. The additional union with *false?* is introduced so that the process of calling recursively would itself "cost" an edge in the tree. A remark related to this matter appears in Section 7.4.

The computation tree  $ct(\alpha, J)$  is obtained from  $pct(\alpha, J)$  by deleting some of the failure nodes as described in Section 5.2, and  $loop_\alpha$  and  $fail_\alpha$  are also defined precisely as in that section.

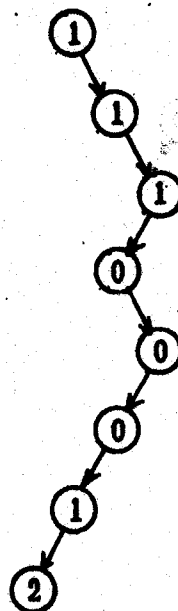
*Examples:* In the following,  $J$  is some state in  $N$  for which  $y_J=0$ , and in the diagrams we let  $i$  stand for  $[i/y]J$ .

(a) Take  $\alpha$  to be the program  $((y=0?;y\leftarrow y+1) \cup (y\neq 0?;y\leftarrow y-1;X;y\leftarrow y+1))^*(J)$ .

$pct(\alpha, [1/y]J)$

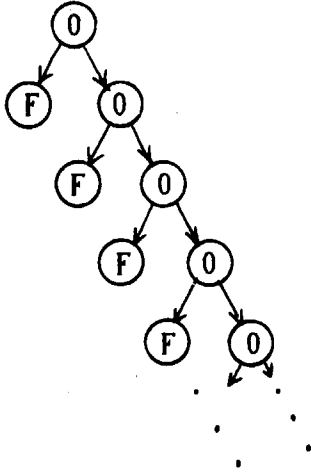


$ct(\alpha, [1/y]J)$

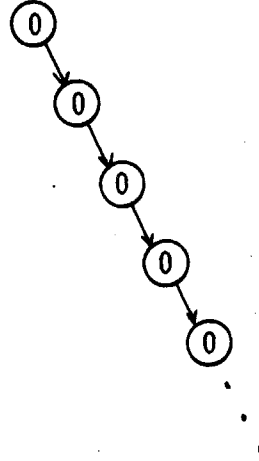


(b) Take  $\alpha$  to be  $(X)^*(f)$ ; this is the recursive program which calls itself recursively "for ever".

$pct(\alpha, J)$



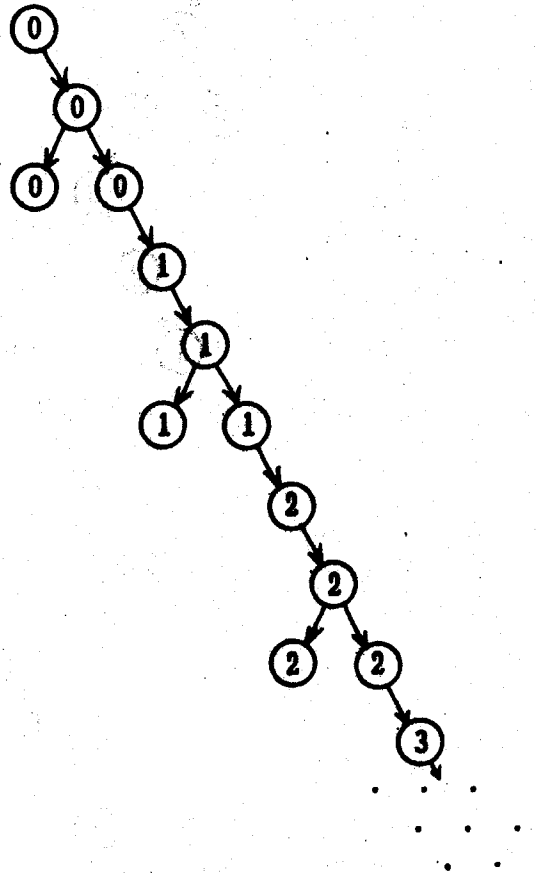
$ct(\alpha, J)$





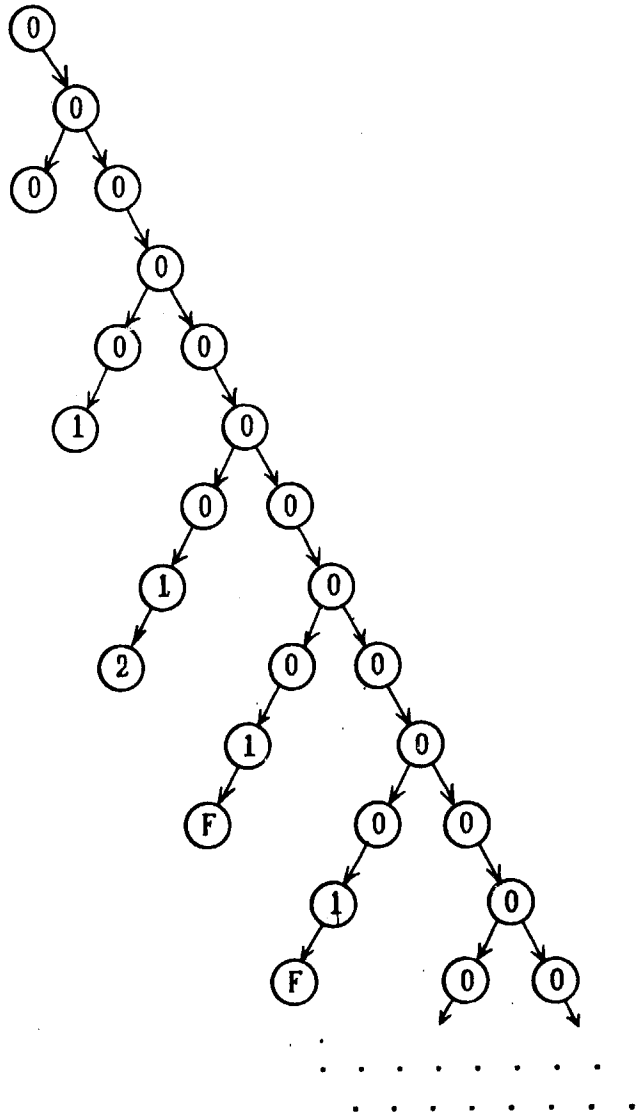
(c) We now show how the two different translations of  $\alpha^*$  into CF formulae (both of which gave rise to the same binary relation; cf. Lemma 4.2) give rise to two different computation trees. As we formally state below, it is only the first of these two which gives rise to trees which, as far as  $loop_\alpha$  and  $fall_\alpha$  are concerned, are identical to those for  $\alpha^*$ . In all cases we supply a program  $\alpha$  and the tree  $ct(\alpha, 1)$ .

$(true? \cup (y \leftarrow y+1; X))^*(f)$

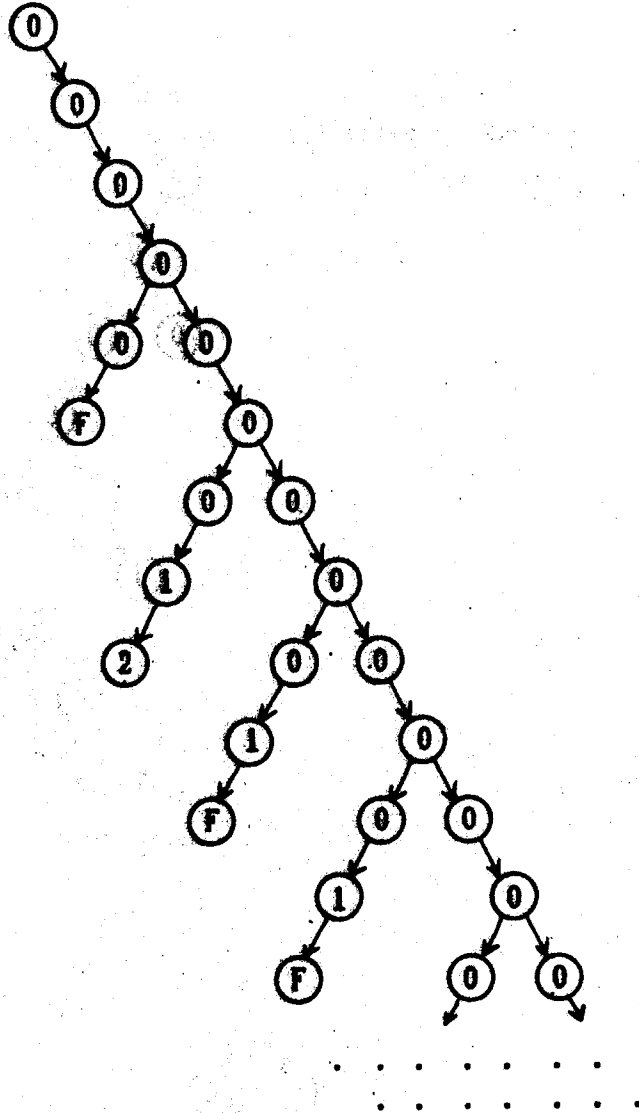




$(true? \cup (X; y < 2?; y \leftarrow y + 1))^*(f)$



$(true? \cup (X; y < 2?; y + y + 1))^*(f); y = 2?$



Here too we have:

**Theorem 7.1:** For every  $\alpha \in CF$ ,  $(J, J) \in m(\alpha)$  iff  $ct(\alpha, J)$  has a leaf labeled  $J$ .

*Proof:* Compare  $pci(\tau^*(f), J) = pci(false? \cup \tau(\tau^*(f)), J)$  with the observation that  $m(\tau^*(f)) = m(false? \cup \tau(\tau^*(f)))$ . ■

The following Theorem substantiates the remark we made in example (c):

**Theorem 7.2:** For any  $\alpha \in RC$  and  $J \in \Gamma$ , denote by  $\alpha'$  the program (in CF) obtained by replacing every appearance of a subprogram of the form  $\beta^*$  in  $\alpha$ , by  $(true? \cup (\beta; X))^*(f)$ . Then we have

$$\begin{aligned} & \models (loop_{\alpha} = loop_{\alpha'}), \\ \text{and} & \quad \models (fail_{\alpha} = fail_{\alpha'}). \end{aligned}$$

*Proof:* The claim follows by observing that  $pc(\beta^*, J) = pc((true? \cup \beta; \beta^*), J)$  and  $pc((true? \cup (\beta; X))^*(f), J) = pc((false? \cup (true? \cup (\beta; true? \cup (\beta; X))^*(f))))(J)$ , but that the failure node due to the *false?* in the latter is always deleted in the process of constructing  $ct$ . ■

Note that for any  $\alpha \in CF$  and  $J \in \Gamma$ ,  $\alpha(\alpha, J)$  is again a tree of finite outdegree, so that Koenig's Lemma [31] can be applied.

## 7.2 Diverging and Failing in CFDL.

As in Section 6.1, we are interested in providing, for any  $\alpha \in CF$ , CFDL-wffs  $P_{\alpha}$  and  $R_{\alpha}$  such that we have  $\models (P_{\alpha} = loop_{\alpha})$  and  $\models (R_{\alpha} = fail_{\alpha})$ . In both cases we will need tools similar to those developed for the corresponding results in Sections 6.1.1 and 6.1.2, but here a brand new problem occurs, the solution of which requires defining the formula  $along(\tau, Q)$  asserting, for a term  $\tau(X)$  and formula  $Q$ , that  $Q$  is true at some point just preceding a recursive call to  $\tau$  during a legal execution of  $\tau^*(f)$ .

### 7.2.1 Expressing $loop_{\alpha}$ in CFDL.

We are looking for a characterization of  $loop_{\tau^*(f)}$  analogous to that of  $loop_{\alpha^*}$  in Theorem 6.2, in order to try to use it, together with Lemma 6.1 and a possible analogue of Theorem 6.3, for obtaining our result.

Recall that according to Theorem 6.2 a divergence in  $\alpha^*$  is due either to a local divergence, i.e. a divergence in some reachable execution of  $\alpha$ , or to a global one, i.e. being able to execute  $\alpha$ 's for ever. The former possibility is  $\langle \alpha^* \rangle loop_{\alpha}$ , which, as is

implicit in the proof of Theorem 6.12, can be written  $\exists \text{loop}_{\alpha} n$ , and the latter is  $\forall n \langle \alpha^n \rangle \text{true}$ . So we can write

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

Characterizing  $\text{loop}_{\tau^*}(\mathcal{J})$  is similar; here a local divergence is a divergence "inside" some application of a reachable  $\tau$ , and can be expressed by  $\exists \text{loop}_{\tau} n(\text{false?})$ .

(Note that this still does not solve the problem of expressing local diverging in GFDL; we deal with this question later.) Global diverging, on the other hand, is more subtle. Here we want to express the possibility of being able to "apply  $\tau$  for ever", which amounts to being able to "proceed infinitely deep into the recursion".

In order to capture this notion we restrict ourselves in this chapter to universes  $U$  in which the domain has at least two distinct elements and in which two fixed variables have these two elements as values. We will therefore use the symbols  $a$  and  $b$  freely as two variables with distinct values.

We now define, for any term  $\tau$ , the term  $\tau'$  which allows "skipping" tests, recursive calls to  $\tau$ , and other recursive constructs, but forces any such skip to be recorded in a new variable  $x$ . Formally, given  $\tau(X)$ , let  $x, y \text{ var}(\tau)$  be two variables, and let  $\tau'(X)$  be  $\tau(X)$  with every appearance of a subterm  $\alpha$  of one of the forms  $X$ ,  $P?$  or  $\tau^m(\mathcal{J})$  replaced by  $(\alpha \cup x \leftarrow b)$ . Also define

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

For any  $n \geq 0$  denote the program  $x \leftarrow a; y \leftarrow a; \tau'^n(\sigma)$  by  $\tau_n$ . We can now present our characterization of  $\text{loop}_{\tau^*}(\mathcal{J})$ :

**Theorem 7.3:** For any  $\alpha \in \text{CF}$ ,

$$\models (\text{loop}_{\tau^*}(\mathcal{J}) \equiv (\exists \text{loop}_{\tau} n(\text{false?}) \vee \forall n \langle \tau_n \rangle y = b)).$$

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

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

$$S(0, \tau, J) = \{J\},$$

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

where  $V$  is the set of states  $J$  such that the process of constructing  $cl(\tau(x+x; \tau^*(f)), J)$  for  $x \notin var(\tau)$  requires constructing  $cl(x+x, J)$ . In other words,  $V$  is the set of states which execution of  $\tau^*(f)$  can reach just prior to calling  $\tau$  recursively at level 1.

Certainly if for some  $i$  we have  $J \in S(i, \tau, J)$ , then  $J$  labels some node in  $cl(\tau^*(f), J)$ , and furthermore the path from the root of  $cl(\tau^*(f), J)$  to that node is of length at least  $i$ . (Note that this would not be the case if we were to define  $pcr(\tau^*(f), J)$  to be  $pcr(\tau(\tau^*(f)), J)$ .)

Assume now that  $\exists n \langle \tau_n \rangle y = b$ . We show that for any  $i \geq 0$  we have  $S(i, \tau, J) \neq \emptyset$ , and thus  $cl(\tau^*(f), J)$  has paths of arbitrary length and is therefore, by Koenig's Lemma, infinite. (Note that the assumption  $\exists n \langle \tau_n \rangle y = b$  is sufficient, so that  $\forall n$  can be replaced by  $\exists n$  in the statement of the Theorem.) Indeed, for any such  $i$ , by assumption, we have  $\exists \tau_i \langle \tau_i \rangle y = b$ , or  $\exists x \langle x \leftarrow a; y \leftarrow a; \tau^i(f) \rangle y = b$ , so that there exists a finite path  $p$  in  $cl(\tau_i, J)$ , 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

$$(J, [a/x]J, [a/y][a/x]J, J_0, \dots, J_k)$$

where  $y_{J_k} = b$ . Let  $i$  be the least integer  $j$  such that  $y_{J_j} = b$ . 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  $j \geq 0$  we have  $x_{J_j} = a$ , 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  $J_j$  can be thought of as being a simulation, in  $\tau^*(\sigma)$ , of a path from the root to the *false?* in  $cl(\tau^i(\text{false?}), J)$ . Consequently, we have  $J_j \in S(i, \tau, J)$ . This completes the proof of one direction of the theorem.

Conversely, assume now that  $\exists \text{loop}_{\tau^*(f)}$  holds and that for all  $n \geq 0$  we have  $\exists \text{loop}_{\tau^n(\text{false?})}$ . Consider the infinite sequence  $s$  of successive labels of the nodes of an infinite path from the root in  $cl(\tau^*(f), J)$ . It is easy to see that by the second hypothesis, there must exist a subsequence of  $s$ , say  $(J_0, J_1, \dots)$ , such that for every  $i$  we have  $J_i \in S(i, \tau, J)$  and such that  $J_i$  corresponds to the first time in  $s$  that "depth  $i$ " of recursion was reached. We show that  $\exists \tau \langle \tau \rangle y = b$  holds for every  $i$  by giving an algorithm

for executing  $\tau_i$  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  $J_1$ , 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  $J_1$ , reaching  $J_1$  in  $s$  corresponds to reaching  $\sigma$  for the first time in  $\tau^i(\sigma)$ . Thus, we have reached  $\sigma$  with  $y_{J_1} = a$  and  $x_{J_1} = a$  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  $\sigma$  any subsequent arrival at  $\sigma$  will not change the value of  $y$ , and since  $y \notin \text{var}(\tau)$ , this value is not changed by any other part of the rest of the execution. Thus,  $y = b$  upon termination.

We are now interested in providing ways for expressing the two disjuncts in the statement of Theorem 7.3 by CFDL-wffs.

For dealing with the left disjunct, consider the set  $\Sigma_f^T = \bigcup_{i=0}^{\infty} S(i, \tau, J)$  which, intuitively, is the set of states which label those nodes in  $\alpha(\tau^*(f), J)$  which correspond to points just prior to a recursive call to  $\tau$ . Assume we have defined, for any CFDL-wff  $Q$  and term  $\tau(X)$ , a formula  $\text{along}(\tau, Q)$  such that

$$J \text{ along}(\tau, Q) \text{ iff } (\exists f \in \Sigma_f^T)(J \vdash Q),$$

i.e.  $J \text{ along}(\tau, Q)$  holds iff  $Q$  is true immediately prior to some reachable recursive call to  $\tau$  in an execution of  $\tau^*(f)$  starting in state  $J$ . Assume also that we have defined, for every program  $\alpha \in CF$  and term  $\tau(X)$ , a formula  $\text{Intloop}_{\tau, \alpha}$  such that, intuitively,  $J \vdash \text{Intloop}_{\tau, \alpha}$  holds iff there is a divergence in  $\alpha(\tau(\alpha), J)$  which is due to the  $\tau$  part and not to the  $\alpha$  part (i.e. the divergence came from some recursive construct appearing in  $\tau(X)$ ).

It is quite clear that  $J \vdash \text{Intloop}_{\tau, n}(\text{false})$  holds iff at some state  $J$  in the execution of  $\tau^*(f)$  just prior to a recursive call to  $\tau$ , it is the case that there is a divergence in  $\alpha(\tau(\tau^*(f)), J)$  which is due to the first  $\tau$  and not to the inner  $\tau^*(f)$ . In other words  $J \text{ along}(\tau, \text{Intloop}_{\tau, \tau^*(f)})$ .

Now we proceed to define these concepts, and then observe that, together with Lemma 6.1 and Theorem 7.3, they give rise to CFDL-wffs. Finally, we state the claim made in the previous paragraph as a theorem.



For any  $\alpha, \beta, \beta' \in CF$  and terms  $\tau_1(X)$  and  $\tau_2(X)$  define

$$\begin{aligned} lp_{X,\alpha} &=_{df} \text{false}, \\ lp_{\beta,\alpha} &=_{df} \text{loop}_{\beta}, \\ lp_{\beta;X,\alpha} &=_{df} \text{loop}_{\beta}, \\ lp_{X;\beta,\alpha} &=_{df} \langle \alpha \rangle \text{loop}_{\beta}, \\ lp_{\beta;X;\beta',\alpha} &=_{df} (\text{loop}_{\beta} \vee \langle \beta' \rangle lp_{X;\beta',\alpha}), \\ lp_{\tau_1 \vee \tau_2, \alpha} &=_{df} (lp_{\tau_1, \alpha} \vee lp_{\tau_2, \alpha}). \end{aligned}$$

Now for defining  $\text{along}(\tau, Q)$  we use tricks similar to those used in constructing  $\tau'$  and  $\sigma$  for Theorem 7.3. Given  $\tau(X)$ , let  $x, y \notin \text{var}(\tau)$  be two variables, let  $Z = \text{var}(\tau)$  and let  $Z'$  be a vector of disjoint primed versions of the variables in  $\text{var}(\tau)$  (see Chapter 4; in particular  $x, y \notin \text{var}(\tau)$ ). Define  $\tau''(X)$  to be  $\tau(X)$  with every appearance of a subprogram  $\alpha$  of the form  $P?$  or  $\tau''(f)$  replaced by  $\langle \alpha \cup x \leftarrow b \rangle$ , 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  $\text{along}(\tau, Q)$  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 7.3, 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.

From these observations, together with the remarks preceding the construction of  $lp_{\tau, \alpha}$  and  $\text{along}(\tau, Q)$ , we obtain:

**Theorem 7.4:** For every term  $\tau(X)$ , we have

$$\vdash (\exists \text{loop}_{\tau} \langle \text{false?} \rangle = \text{along}(\tau, lp_{\tau, \tau''(f)})).$$

Now observe that the definition of  $\text{along}(\tau, \text{lp}_{\tau, \tau^*(f)})$  involves only CFDL-wffs and constructs of the form  $\text{loop}_{\alpha}$  where  $\alpha$  includes only subprograms appearing in  $\tau(X)$ . We do not know how to deal with the right hand disjunct of Theorem 7.3, so that we have:

*Open Problem:* Is it the case that for every  $\alpha \in \text{CF}$ , term  $\tau(X)$  and L-wff  $P$  there exists a CFDL-wff  $Q$  such that  $\models (Q \equiv \exists_{\alpha} \langle \tau^*(\alpha) \rangle P)$ .

An affirmative answer to this question would imply, together with Lemma 6.1 and Theorems 7.3 and 7.4, that for CF  $\text{loop}_{\alpha}$  is expressible in CFDL.

### 7.2.2 Expressing $\text{fail}_{\alpha}$ in CFDL.

We have been unable to find an elegant and reasonably natural algorithm for constructing, given  $\alpha \in \text{CF}$ , the CFDL-wff  $R_{\alpha}$  such that  $\models (R_{\alpha} \equiv \text{fail}_{\alpha})$  holds. We can show, though, that such an  $R_{\alpha}$  exists by means of a tedious and uninteresting case analysis. The difficulty was in finding a CFDL-wff  $f_{\tau}$  such that we have

$$\models (\text{fail}_{\alpha} \equiv \text{along}(\tau, f_{\tau})).$$

$\models f_{\tau}$  is to hold whenever there is a failure in  $\text{cl}(\tau(\tau^*(f)), J)$  due to  $\tau$  (i.e. the failure does not appear in  $\text{cl}(\tau^*(f), J)$  for any  $f \in S(1, \tau, J)$ ). The difficulties, similar to those of Section 6.1.2, are when parts of  $\tau$  are tests.

We pose to the reader the interesting problem of designing a useful sublanguage of CF for total-correctness oriented reasoning, which would be to CF what the guarded commands language GC is to RC. This programming language should have the pleasant property that for it  $f_{\tau}$  can be expressed easily and naturally. Then it would be interesting to try and find concise rules for constructing some or all of the four notions of total correctness or weakest preconditions described in Chapter 5, similar to those provided by Dijkstra [14] for GC (cf. Section 5.5).

### 7.3 CFDL Augmented with $\text{loop}_{\alpha}$ (CFDL<sup>+</sup>).

In this section we augment CFDL with  $\text{loop}_{\alpha}$  and refer to the resulting logic as CFDL<sup>+</sup>. Although there seems to be no reason to abbreviate  $\langle \alpha \rangle P \vee \text{loop}_{\alpha}$  to  $\langle \alpha \rangle^+ P$ , we

will do so in order to be consistent with the treatment of  $DL^+$  in Section 6.2. The virtues of augmenting CFDL with  $loop_{\alpha}$  are those described for DL vs.  $DL^+$  at the beginning of Section 6.2, with the additional point that it might turn out that for CF  $loop_{\alpha}$  is not expressible in CFDL, and so we would have  $CFDL < CFDL^+$ , in which case the augmentation is proper in the sense of obtaining strictly more expressive power. The axiomatization of  $CFDL^+$  which we provide in Section 7.3.2 is not quite as natural looking as that of Section 6.2.2 for  $DL^+$ . We are of the opinion that a search for a clean new formalism for reasoning naturally about recursive programs (perhaps taking along  $(\tau, Q)$  as a primitive) might be worthwhile, although we are somewhat doubtful about the chances of it bringing about a significant improvement.

### 7.3.1 Definitions.

The definition of  $CFDL^+$  is similar to  $DL^+$ , taking the definitions of the basic concepts from DL and adding:

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

We abbreviate as in Section 6.2.1 and define the semantics inductively using the definition of  $loop_{\alpha}$  of Section 7.1.

### 7.3.2 Axiomatization of $CFDL^+$ .

The basis of our axiomatization is Theorem 7.3 which we can now rephrase as:

$$\models \langle \tau^*(f) \rangle^* \text{false} = (\exists n \langle \tau^n(\text{false?}) \rangle^* \text{false} \vee \forall n \langle \tau_n \rangle y=b)$$

$$\text{and } \models [\langle \tau^*(f) \rangle]^* \text{true} = (\forall n [\langle \tau^n(\text{false?}) \rangle]^* \text{true} \wedge \exists n [\langle \tau_n \rangle] y \neq b).$$

Here  $\tau_n$  is the program  $(x \leftarrow a; y \leftarrow a; \tau^n(\sigma))$  where  $\tau$  and  $\sigma$  were defined preceding Theorem 7.3. Also, in the sequel we use  $\tau^*(X)$  as defined preceding Theorem 7.4.

Our axiomatization here too will be of an extension  $CFDL^+$  which is defined as  $CFDL^+$  but with the programs coming from the set  $CF^*$ . As in Chapter 4, we will be using

the fact that in an arithmetical universe  $A$  there exists, for any  $\alpha \in CF$ , an L-wff  $P$  such that  $P^Z$  expresses  $\alpha$ . The problem that arises is that of defining the tree  $ct(\alpha, J)$  for  $\alpha \in CF'$  (as opposed to  $CF$ ), or alternatively as far as this section is concerned, that of defining  $loop_\alpha$ . We would like it to be the case that for any  $P^Z$ ,  $P-loop_\alpha(P^Z)$  holds. However, for a given  $J \in \Gamma$  it is possible that the set  $\mathcal{J}(P^Z) = \{J \mid (J, J) \in m(P^Z)\}$  is infinite.

One solution is to define  $ct(\alpha, J)$  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

$$ct(P^Z, J) = \{(\lambda, J)\} \cup \{(1, J_1) \mid (J, J_1) \in m(P^Z)\}.$$

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

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

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

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

*Example:* Let  $\alpha: x < x+1$ ,  $P: x < x'$  and  $Z = (x)$ . For any  $J \in \mathbb{N}$  such that  $x_J = 0$  we have:

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

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

and thus  $CT(\alpha; P^Z, J) = \{ \{(\lambda, J), (0, [1/x]J), (00, J)\} \mid x_J > 1 \}$ . ■

Now define  $J \# loop_\alpha$  iff there is an infinite tree in  $CT(\alpha, J)$ .

We remark that either way  $loop_\alpha$  is uniquely defined for  $\alpha \in CF'$ , and that for  $\alpha \in CF$ ,  $CT(\alpha, J) = \{ct(\alpha, J)\}$ .

Let  $A$  be any arithmetical universe, and consider the axiom system  $R^+$  for  $CFDL^+$  defined as  $R$  of Section 4.3 augmented with axioms (O)-(S) of  $P^+$  in Section 6.2.2 and the following axioms and rules:

(In the following,  $P$  and  $Q$  are L-wffs,  $R$  is a  $CFDL^+$ -wff,  $\tau(X)$  is a term,  $x$  and  $y$  are variables  $x, y \notin var(\tau)$ ,  $Z = var(\sigma)$ ,  $V$  is the vector of variables obtained by augmenting  $Z$  with  $x$  and  $y$ , and  $\sigma$ ,  $\tau'$  and  $\tau''$  are as defined above.)

$$(V) \quad [P^Z]^+ true,$$

(W)

$$R \supset ( \langle x \leftarrow a; y \leftarrow a; \tau''(f) \rangle (y=b \wedge \langle Z \leftarrow Z' \rangle \langle \tau(Q^Z) \rangle^+ false) \vee \forall n \langle x \leftarrow a; y \leftarrow a; P(n)^V \rangle_{y=b} ) , \\ P(0, V, V') \supset \langle \sigma \rangle V = V'' , \quad Q(Z, Z') \supset \langle \tau''(f) \rangle Z = Z' , \quad P(n+1, V, V') \supset \langle \tau'(P(n)^V) \rangle V = V'$$

---


$$R \supset \langle \tau''(f) \rangle^+ false$$

(Y)

$$R \supset ( [x \leftarrow a; y \leftarrow a; \tau''(f)] (y \neq b \vee [Z \leftarrow Z'] \langle \tau(Q^Z) \rangle^+ true) \wedge \exists n [x \leftarrow a; y \leftarrow a; P(n)^V]_{y \neq b} ) , \\ V = V' \supset \langle \sigma \rangle P(0, V', V) , \quad Z' = Z \supset \langle \tau''(f) \rangle Q(Z', Z) , \quad V' = V \supset \langle \tau'(P(n)^V) \rangle P(n+1, V', V)$$

---


$$R \supset \langle \tau''(f) \rangle^+ true$$

Provability in  $R^+$  is defined as usual.

**Theorem 7.5 (A-soundness of  $R^+$ ):** For any  $CFDL$ -wff  $P$ , if  $\vdash_{R^+} P$  then  $\vdash_A P$ .

*Proof:* We establish the A-soundness of the additional axiom and rules, and then use Theorems 4.10 and 6.15 to conclude the result.

We show then, that for any L-wffs  $P$  and  $Q$ ,  $CFDL$ -wff  $R$  and term  $\tau(X)$ , with  $x, y, \tau', \tau'', \sigma, Z$  and  $V$  as above, axiom (V) is A-valid, and rules (W) and (Y) preserve A-validity.

(V): By definition.

(W): 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

$$\vdash_A (R \supset (\exists n \text{loop}_{\tau^n}(\text{false?}) \vee \forall n \langle \tau_n \rangle y=b),$$

which, by Theorem 7.3, implies that  $\vdash_A (R \supset \langle \tau^*(f) \rangle^+ \text{false})$ . (Recall that  $\tau_n$  is an abbreviation of  $\langle x+a; y+a; \tau^n(\sigma) \rangle$ .) And indeed, by Theorem 6.3 the premises, other than the first, assert, respectively,  $m(P(0)^V) \subseteq m(\sigma)$ ,  $m(Q^Z) \subseteq m(\tau^*(f))$ , and  $\forall n (m(P(n+1)^V) \subseteq m(\tau^n(P(n)^V)))$ . One can then show, by induction on  $n$  using Lemma 4.4, that  $\forall n (m(P(n)^V) \subseteq m(\tau^n(\sigma)))$ . Consequently, since  $Q^Z$  is "smaller" as a relation than  $\tau^*(f)$  but is divergence-free, one can see that  $\text{loop}_{\tau}(Q^Z)$  implies  $\text{lp}_{\tau, \tau^*(f)}$ , and hence also that  $\text{along}(\tau, \text{loop}_{\tau}(Q^Z))$  implies  $\text{along}(\tau, \text{lp}_{\tau, \tau^*(f)})$ . By Theorem 7.4 the latter is  $\exists n \text{loop}_{\tau^n}(\text{false?})$ . Moreover, since for any  $n$ ,  $P(n)^V$  is "smaller" than  $\tau^n(\sigma)$ , one can see that  $\forall n \langle x+a; y+a; P(n)^V \rangle y=b$  implies  $\forall n \langle \tau_n \rangle y=b$ . Thus, the A-validity of the first premise of rule (W) implies that  $R \supset \text{loop}_{\tau^*(f)}$  is A-valid, and hence we obtain the A-validity of the conclusion. (Y): Dual reasoning to that of (W). ■

The proof of arithmetical completeness of  $\mathbb{R}^+$  follows the framework of similar proofs in the previous chapters. We apply Theorem 3.1 after establishing that its hypotheses hold in this particular case. First we have:

**Theorem 7.6:** L is A-expressive for  $\text{CF}^+\text{DL}^+$ .

*Proof:* Trivial using Theorem 4.1 and Corollary 7.6. ■

Now we prove the basic box<sup>+</sup>- and diamond<sup>+</sup>-completeness results, and then, following our remark, in Section 6.2.2, about a "double functional" version of Theorem 3.1, we obtain our final result.

**Theorem 7.7 (Diamond<sup>+</sup>-completeness Theorem for  $\text{CF}^+\text{DL}^+$ ):** For every  $\alpha \in \text{CF}^+$  and L-wffs R and Q, if  $\vdash_A (R \supset \langle \alpha \rangle^+ Q)$  then  $\vdash_{\mathbb{R}^+} (R \supset \langle \alpha \rangle^+ Q)$ .

*Proof:* As in the proof of Theorem 6.19, it is easy to see that all we need to show is that if  $\vdash_A (R \supset \langle \alpha \rangle^+ \text{false})$  then  $\vdash_{\mathbb{R}^+} (R \supset \langle \alpha \rangle^+ \text{false})$ . This, again, is established by induction on the structure of  $\alpha$ . When  $\alpha$  is of the form  $\tau^*(f)$  for some term  $\tau$  we show the existence of L-wffs Q and P(n) such that the premises of rule (W) are A-valid. Since these premises involve only  $\text{CF}^+\text{DL}$ -wffs and the formula  $\langle \tau(Q^Z) \rangle^+ \text{false}$ , in which the program is of complexity lower than  $\tau^*(f)$ , the result will follow. Indeed, by Theorem 4.1 we can take Q and P(n) to be L-wffs involving, respectively, only variables in Z and V,

and such that  $\vDash_A(Q \equiv \tau^*(f))$  and for all  $n$   $\vDash_A(P(n) \equiv \tau^n(\sigma))$ . All the premises are easily seen to be  $A$ -valid for this choice. ■

**Theorem 7.8 (Box<sup>+</sup>-completeness Theorem for CF'DL<sup>+</sup>):** For every  $\alpha \in CF^+$  and L-wffs  $R$  and  $Q$ , if  $\vDash_A(R \supset [\alpha]^+ Q)$  then  $\vdash_{R^+}(R \supset [\alpha]^+ Q)$ .

*Proof:* As above using rule (Y).  $Q$  and  $P(n)$  are also taken precisely as above. ■

And thus, as remarked, we conclude:

**Theorem 7.9 (Arithmetical Soundness and Completeness for CF'DL<sup>+</sup>):** For every CF'DL<sup>+</sup>-wff  $P$ ,

$$\vDash_A P \quad \text{iff} \quad \vdash_{R^+} P.$$

Appendix E contains a proof of a CF'DL<sup>+</sup>-wff in  $R^+$ .

## 7.4 Language Dependent Diverging and Failing.

In this section, based upon an idea of Meyer [44], we show how it is possible to define notions of diverging and failing which depend, not on any particular definition of computation trees, but solely upon the language generated from, say, the regular expressions. In fact, the new notions are well defined for any "program" consisting of a set of sequences of assignments and tests. An immediate upshot is the fact that these concepts of *language-diverging* and *language-failing* are defined for r.e. programs as well as for regular and context-free ones (see Section 2.5.5). However, the new notions, being independent of the particular expression (or grammar) defining the program, do not coincide precisely with our *loop<sub>α</sub>* and *fail<sub>α</sub>*. The brief technical investigation of this phenomenon which we supply below, sheds some light on the reasons for adopting the seemingly *ad hoc* definitions of computation trees in Sections 5.2 and 7.1.

Let  $\Delta$  be the alphabet consisting of legal assignments and tests in DL. The programs we consider here are subsets of  $\Delta^*$ , i.e. sets of finite strings of assignments and tests. We use  $B, C, \dots$  to denote such programs, with  $\lambda$ , the empty string, denoting the identity program.

Let  $J \in \Gamma$ ,  $B \subseteq \Delta^*$ , and  $a \in B$  such that  $a \neq \lambda$ . We say that  $a$  is *J-good* if we have  $J \vDash \langle \alpha_a \rangle \text{true}$ , where  $\alpha_a$  is the straight-line DL program obtained by inserting ";" between

every two elements in the string  $a$ . Now define  $J\text{-lang-loop}_B$  iff there exists an infinite string  $s$  over  $\Delta$ , every finite prefix of which is a prefix of an  $J$ -good element of  $B$ . Intuitively,  $J\text{-lang-loop}_B$  asserts that it is possible to execute an element of  $B$  and then extend that element repeatedly for ever, executing the extension each time, without ever "leaving"  $B$ .

In order to be able to compare  $\text{lang-loop}$  with  $\text{loop}$  we adopt the standard translation  $T$  of a regular expression into the language (set of strings) it defines. Define  $T: RC \rightarrow 2^{\Delta^*}$  as follows:

$$\begin{aligned} T(x \leftarrow e) &= \{x \leftarrow e\}, \\ T(P?) &= \{P?\}, \\ T(\alpha; \beta) &= \{ab \mid a \in T(\alpha) \wedge b \in T(\beta)\}, \\ T(\alpha \cup \beta) &= T(\alpha) \cup T(\beta), \\ T(\alpha^*) &= (T(\alpha))^*. \end{aligned}$$

We now observe that, contrary to expectation, it is not the case that for all  $\alpha \in RC$  we have  $\models (\text{loop}_\alpha = \text{lang-loop}_{T(\alpha)})$ . This follows from observing that although  $T(\alpha^*) = T(\alpha^{**})$ , and although  $\models \text{loop}_{\alpha^*}$  does not necessarily hold,  $\models \text{loop}_{\alpha^{**}}$  always holds. This situation is perhaps best explained by showing what has to be done to a regular expression, i.e. a program  $\alpha$  in  $RC$ , in order to be able to capture  $\text{loop}_\alpha$  using language-diverging.

For any  $\alpha \in RC$  define  $\alpha'$  to be  $\alpha$  with every subprogram of the form  $\beta^*$  replaced by  $(\text{true?} \cup \beta^*)$ . Thus, we are explicitly adding the fact that "doing nothing" is a legal execution of  $\beta^*$ . In this way, carrying out this degenerated (but nonempty) computation for ever results in a divergence which is captured by the infinite set of strings  $\{\text{true?}^k\}_{k \geq 0}$ . Formally, we have

**Lemma 7.10 (Meyer [44]):** For any  $\alpha \in RC$ ,  $\models (\text{loop}_\alpha = \text{lang-loop}_{T(\alpha')})$ .

Turning now to the concept of failing, we would like to mark those elements of  $a \in B$  which are not  $J$ -good, by pruning them at the point where a test failed and inserting the special indicator  $F$ . Define a mapping  $\psi: \Delta^* \times \Gamma^* \rightarrow (\Delta \cup \{F\})^*$  as follows:



$$\psi(\lambda, J) = \lambda,$$

$$\psi(x \leftarrow e, J) = x \leftarrow e,$$

$$\psi(P?, J) = \begin{cases} \lambda & \text{if } J \models P \\ F & \text{if } J \not\models P \end{cases}$$

$$\psi(a; b, J) = \begin{cases} \psi(a, J)\psi(b, J) & \text{if } J \in J(\alpha_2) \\ \psi(a, J) & \text{if } J(\alpha_2) = \emptyset. \end{cases}$$

It is easy to see that this definition is a unique one. In fact, for any  $a$  and  $J$ ,  $\psi(a, J)$  includes only assignments, and possibly one  $F$  as the last element in the string.

Language-failing is now defined as follows:  $J \models \text{lang-fail}_B$  iff there exists  $a \in B$  such that  $\psi(a, J) = bF$ , and such that for no  $c \in B$  is it the case that  $\psi(c, J) = bd$  where  $d \neq F$ . The intuition is that  $B$  includes a language-failure in state  $J$  if one can execute a sequence of instructions  $a \in B$  starting in state  $J$ , and reach a false test without being able to continue from that point in some other sequence in  $B$  (i.e. no immediate alternative).

Here too, it is not the case that  $\models(\text{fail}_\alpha \equiv \text{lang-fail}_T(\alpha))$ . The counter example being  $\alpha: (x \leftarrow e \cup (\text{false?} \cup \text{false?}))$  for which we have  $\models \text{fail}_\alpha$  but  $\not\models \text{lang-fail}_T(\alpha)$ . We proceed similarly:

For every  $\alpha \in RC$ , define  $\alpha''$  to be  $\alpha$  with every subprogram of the form  $\beta \cup \gamma$  replaced by  $(x \leftarrow x; \beta \cup \gamma \leftarrow \gamma)$ , for some  $x, y \notin \text{var}(\alpha)$ . Thus, we are marking the fact that we have executed a union and have gone left or right.

**Lemma 7.11:** For any  $\alpha \in RC$ ,  $\models(\text{fail}_\alpha \equiv \text{lang-fail}_T(\alpha''))$ .

A similar treatment of the recursive programming language CF can be carried out. Here the counter example to  $\models(\text{loop}_\alpha \equiv \text{lang-loop}_T(\alpha))$ , with  $T$  extended in the standard way to context free grammars, is the program  $\alpha: (X)^*(f)$  for which we have  $\models \text{loop}_\alpha$  but not  $\models \text{lang-loop}_T(\alpha)$  since  $T(\alpha) = \emptyset$ . The coding trick needed here in order to capture  $\text{loop}_\alpha$  by asserting  $\text{lang-loop}_T(\alpha''')$  is to take  $\alpha'''$  to

be  $\alpha$  with every program variable  $X$  in a subprogram of the form  $t^*(f)$  replaced by  $(true?;X)$ . Thus, we are marking the fact that a recursive call "costs" a unit.

This particular direction of defining "grammar independent" notions seems to justify careful investigation. It is appealing in part because it does not assume any extension of the standard definitions of such operators as  $*$  when applied to programs. Its drawback, however, seems to be in the fact that in order to capture such (in our opinion highly intuitive and natural) concepts as *loop* and *tail*, one needs to invent a form of encoding from which, in effect, the original definition (in this case the computation trees) can be reconstructed.

## 8. Conclusion and Directions for Future Work.

The following seem to be the main contributions of this thesis:

- (1) Provision of a comprehensive and rigorous description of work on dynamic logic.
- (2) Introduction of the notion of arithmetical axiomatization and provision of concise arithmetically complete axiom systems for a variety of logics.
- (3) Introduction of the notions of diverging and failing and, with their aid, clarification of the concepts of total correctness and weakest preconditions.
- (4) Provision of an analogy between iteration and recursion, giving rise to a clean axiomatization of recursive dynamic logic, and exposing the difficulties involved in reasoning about the diverging and failing of recursive programs.

There is still a lot of work to be done. Most of the open problems scattered throughout the thesis are to do with comparative power of expression. It seems that some of them will turn out to be quite easy, and we believe that as each is solved more insight will be gained, thus easing the task of solving the others.

The main directions, directly related to this thesis, in which we would recommend that further work be done are:

- (1) *Recursive programs:* We feel that there ought to be a more natural way to reason about recursion. As is quite evident from our work on  $\alpha^*$ , the primitives of dynamic logic are not only adequate for expressing interesting properties of iterative programs, but also enable the reasoning about these properties to be carried out inductively in a structured manner. For some properties  $P$  of programs, a natural way in which to prove them of  $\alpha^*$ , is simply to prove them for "every  $\alpha$  in  $\alpha^{**}$ " by proving  $[\alpha^*]P(\alpha)$ . Thus the problem is reduced one level. This is the essence of the rules for  $\alpha^*$  in our various axiom systems. For recursive programs the situation seems to be different. Here a one-level reduction of the problem of showing a property to hold of  $\tau^*(f)$ , is to show that

it holds of  $\tau$  when  $\tau^*(f)$  is "plugged in". Thus, the *along*( $\tau, Q$ ) construct of Section 7.2 seems to be an important notion. And so, although the primitives of recursive dynamic logic still capture some of the properties that we are ultimately interested in (and so they should not be eliminated), the methods for proving these properties and for expressing other properties of interest (such as *fail* $_{\tau^*(f)}$ ) seem to be in need of extra tools.

(2) *Computation trees*: These should be further investigated carefully, perhaps having in mind the need for making the tree a "fair" description of the computation by assigning costs to assignments and tests, but not to "dummy" edges such as those which correspond to the "go left" and "go right" of the union operator. These trees then could serve as the basis for carrying out an analysis of the efficiency of algorithms, with applications to program optimization etc. Our own construction of  $\alpha(\alpha, \beta)$  in Sections 5.2 and 7.1 was strongly influenced by our interest in diverging and failing. Also, a cleaner definition of failing might be worth looking for. Such a definition should not be made to falsify the main results of Chapters 5-7, but could, perhaps if coupled with an appropriate modification of the definition of the trees, make the reasoning about *fail* $_{\alpha; \beta}$  and *fail* $_{\tau^*(f)}$  a lot easier.

(3) *Parallel programs*: Other interesting primitives, which would perhaps enable natural reasoning about parallel programs, are being investigated on the propositional level and some results have already been established (see eg. Pratt [53] and Parikh [49]). These primitives include for example "throughout a P holds" and "sometimes during a P holds". It would be interesting to investigate the first order versions of logics which include these primitives, and then to try constructing natural arithmetical axiomatizations of them. It seems that a clean overall treatment of the problem of reasoning about programs that run in parallel, in the spirit of the work described in this thesis for sequential programs, is yet to be carried out.

## Appendix A: Relational Characterization of EPDL.

We show that EPDL of Section 1.1.1 is embedded in a simple algebra of relations which employs only two operations: conventional relational composition ( $\circ$ ), and a new unary operation on relations, *minus* ( $-$ ). We point to some questions about our relational algebra which seem to justify further research.

Since EPDL does not involve operations on programs ( $;$ ,  $\cup$ ,  $*$ ) this appendix can be viewed therefore as providing a Boolean-algebra like abstraction of a modal logic in which there are possibly many modalities.

Given a set of symbols  $\tau$  including one special symbol  $\theta$  we define the set  $\Psi(\tau)$  of expressions of the relational algebra over  $\tau$  as follows.

- (1) All elements of  $\tau$  are in  $\Psi(\tau)$ ,
- (2) For every  $e$  and  $f$  in  $\Psi(\tau)$ ,  $(e \circ f)$  and  $-e$  are in  $\Psi(\tau)$ .

An interpretation  $I$  of  $\Psi(\tau)$  is a pair  $(V, r)$  where  $V$  is a nonempty set and  $r: \tau \rightarrow 2^{V \times V}$ , such that  $r(\theta) = \emptyset$ .

$r$  is extended to the set of expressions  $\Psi(\tau)$  by

$$\begin{aligned} r(e \circ f) &= r(e) \circ r(f) = \{(s, t) \mid (\exists u) \{(s, u) \in r(e) \text{ and } (u, t) \in r(f)\}\}, \\ r(-e) &= -r(e) = \{(s, t) \mid (\forall t) \{(s, t) \notin r(e)\}\}. \end{aligned}$$

Thus, the minus operator ( $-$ ) connects  $s$  to itself iff  $s$  was connected to no element of  $V$  in the original relation.

*Lemma A.1:* The set  $-\Psi(\tau) = \{-e \mid e \in \Psi(\tau)\}$  is a Boolean algebra with  $\circ$  and  $-$  acting as intersection and complement respectively.

*Proof:* It is easy to show that the standard postulates for a Boolean algebra are satisfied with  $0 = \theta$  and  $1 = -\theta$ . ■

We now define a syntactic translation function from the set of EPDL-wffs to the set of expressions of the relational algebra over the atomic symbols; in other words,  $f: \text{EPDL} \rightarrow \Psi(\text{AF} \cup \text{AP})$ . For esthetic reasons we take EPDL as though it was defined using

$P \supset Q$  and  $[a]P$  instead of  $P \vee Q$  and  $\langle a \rangle P$ . The latter are now to be regarded as abbreviations in the obvious way.

- (1) For every  $p \in AF$ ,  $r(p) = p$ ,
- (2) For every  $a \in AP$  and EPDL-wffs  $P$  and  $Q$ ,
 
$$r(P \supset Q) = \neg(r(P) \cdot \neg r(Q)),$$

$$r(\neg P) = \neg r(P),$$

$$r([a]P) = \neg(a \cdot \neg r(P)).$$

Given a structure  $S = (W, \pi, m)$  for EPDL, define the interpretation  $I_S$  of  $\Psi(AF \cup AP)$  to be  $I_S = (W, r)$ , where

$$\begin{array}{ll} r(p) = m(p?) & \text{for } p \in AF, \\ \text{and } r(a) = m(a) & \text{for } a \in AP. \end{array}$$

The connection between EPDL and  $\Psi(AF \cup AP)$  is captured by the following theorem.

**Theorem A.2:** For every EPDL-wff  $P$ ,  $r(r(P)) = m(P?)$ .

*Proof:* By induction on  $P$ . For  $P$  an atomic formula  $p \in AF$ , we have  $r(r(p)) = r(p) = m(p?)$  by definition.

Consider  $P$  of the form  $R \supset Q$ . Assume  $(s, s) \in \pi(R \supset Q?)$ , so that in fact if  $s \in \pi(R)$  then  $s \in \pi(Q)$ . We show that  $(s, s) \in r(\neg(r(R) \cdot \neg r(Q)))$ , or that  $(\forall t)((s, t) \notin r(r(R) \cdot \neg r(Q)))$ . Indeed, if for some  $t$  we had  $(s, t) \in r(r(R) \cdot \neg r(Q))$ , then there would exist  $u$  such that  $(s, u) \in r(r(R))$  and  $(u, t) \in r(\neg r(Q))$ . By the inductive hypothesis  $r(r(P)) = \{(s, s) \mid s \in \pi(R)\}$ , so that  $u = s$  and  $s \in \pi(R)$ . But  $(s, t) \in r(\neg r(Q)) = \neg r(r(Q))$  means that  $t = s$  and that furthermore  $(\forall v)((s, v) \notin r(r(Q)))$ . In particular, it is impossible that  $(s, s) \in r(r(Q))$ , or by the hypothesis that  $s \in \pi(Q)$ . Contradiction.

Conversely, assume that  $(s, s) \in r(\neg(r(R) \cdot \neg r(Q)))$  and  $s \in \pi(R)$ . By the inductive hypothesis  $(s, s) \in r(r(R))$ . We will derive a contradiction from the additional assumption that  $(s, s) \notin r(r(Q))$ . By assumption  $(s, s) \notin r(r(R) \cdot \neg r(Q))$ , so in particular it is impossible that  $(s, s) \in r(r(Q))$ , which contradicts  $(s, s) \notin r(r(Q))$ . When  $P$  is of the form  $[a]R$ , the proof follows that of the previous case (with  $m(a)$  replacing  $\{(s, s) \mid s \in \pi(R)\}$  and with appropriate modifications), and the case when  $P$  is of the form  $\neg R$  is similar. We omit both. ■

Thus, what we have essentially done is to view  $P \supset Q$  as a special case of  $[a]Q$ , namely where  $a$  is  $P?$ , and thus we have been able to capture the structure of this most

elementary logic of programs in an algebra of relations which employs only two operations. Theorem A.2 shows how to embed EPDL in this algebra  $\Psi$ .

Note that, with notation slightly relaxed and omitting the argument to  $\Psi$ , we have

$$f(\text{EPDL}) \subseteq \{-\Psi \cup \text{AF}\} \subseteq \Psi.$$

Both inclusions are strict; for general  $a \in \text{AP}$  there is no EPDL-wff  $P$  such that  $r(-(a \circ a)) = m(P?)$ , and also there is no expression in  $\{-\Psi \cup \text{AF}\}$  corresponding to  $a$ . An obvious interesting problem, then, would be to investigate the relationship between  $\Psi$  and  $f(\text{EPDL})$ . For example, what is the complexity of deciding *diagonality* in  $-\Psi$ ; i.e. how hard is it to decide, for arbitrary  $e \in -\Psi$ , whether for every interpretation  $r(e) = \{(s,s) \mid s \in W\}$ ? We know that validity in EPDL, and hence diagonality in  $f(\text{EPDL})$ , is decidable. Is this true in  $-\Psi$ ?

Another possible direction to go would involve investigating "abstract" relational algebras; i.e. is it possible to give a finite set of postulates that a triple  $(K, b, u)$  is to satisfy in order for  $b$  and  $u$  to act like  $\circ$  and  $-$ , where  $K$  is a set of binary relations over some arbitrary set and  $b$  and  $u$  are binary and unary operations on  $K$  respectively. One of those postulates, in line with Lemma A.1, would be that  $(u(K), b, u)$  is a Boolean algebra. What happens when  $K$  is merely assumed to be any set? Such *representation theorems* would seem to be of considerable interest.

### Appendix B: Example of a Proof of a DL-off in $\mathcal{P}$ .

We sketch the highlights of a proof in  $\mathcal{P}$  of the total correctness of the (deterministic) program computing McCarthy's celebrated 91-function (see [40]).

We assume the universe  $N$  of pure arithmetic, and use standard symbols such as  $<$  etc. as abbreviations for the obvious first order formulae they stand for. We do not refer to rules in  $\mathcal{P}$  other than those we use for proving  $\langle \gamma^* \rangle$  below. McCarthy [40] showed that the least fixpoint of the recursive definition

$$f(z) = \text{if } z > 100 \text{ then } z - 10 \text{ else } f(f(z + 11))$$

is the function

$$f(z) = \begin{cases} z - 10 & z > 100 \\ 91 & 100 \geq z \end{cases}$$

We consider an iterative version of this recursive definition, in the form of the following regular program  $\gamma^*$ , and show that indeed for inputs  $z$  such that  $100 \geq z$ ,  $\gamma^*$  computes  $f(z) = 91$ , so that  $z = 101$  upon termination.

Define

$$\begin{aligned} \alpha: & 100 \geq z; z \leftarrow z + 11; y \leftarrow y + 1, \\ \beta: & z \leftarrow z - 10; y \leftarrow y - 1, \\ \gamma: & \alpha^*; (100 < z \wedge y = 1)?; \beta. \end{aligned}$$

We prove the  $N$ -validity of

$$(101 \geq z \wedge y = 1) \supset \langle \gamma^* \rangle (z = 101 \wedge y = 1),$$

by defining the convergent  $P(n)$  as follows:

$$P(n): \quad y > 0 \wedge z > 0 \wedge 111 \geq z \wedge n = 90 - z + 11y.$$

(Note:  $P(n)$  is in fact the arithmetical equivalent of  $\langle \gamma^* \rangle (z = 101 \wedge y = 1)$ .)

We prove (in  $\mathcal{P}$ ) the following three formulae, and then an application of the derived rule (J) gives the conclusion:



$$\begin{aligned}
 (*) \quad & (101 \geq z \wedge y=1) \supset \exists n P(n), \\
 & P(n+1) \supset \langle \gamma \rangle P(n), \\
 & P(0) \supset (z=101 \wedge y=1).
 \end{aligned}$$

The first and third of these can easily be seen to be axioms in (B) (i.e.  $N$ -valid L-wffs). We prove the second, (\*).

Abbreviate	$100 < z \wedge P(n)$	to	$P_1(n),$
	$100 \geq z \wedge z \geq 90 \wedge P(n)$	to	$P_2(n),$
and	$z < 90 \wedge P(n)$	to	$P_3(n).$

Certainly we have that the following is  $N$ -valid and hence an axiom:

$$(**) \quad (P_1(n) \vee P_2(n) \vee P_3(n)) \equiv P(n),$$

and so we prove for  $i=1,2,3$ , that

$$P_i(n+1) \supset \langle \gamma \rangle P(n)$$

and use (\*\*) to conclude (\*). We omit the cases  $i=1$  and  $i=2$  which are reasonably straightforward. For  $i=3$  it is sufficient to prove

$$P_3(n+1) \supset \langle \alpha^* \rangle (1 < y \wedge 100 < z \wedge 121 \geq z \wedge n = 89 - z + 11y).$$

We will actually prove

$$P_3(n+1) \supset \langle \alpha^*; \alpha; \alpha \rangle (1 < y \wedge 100 < z \wedge 121 \geq z \wedge n = 89 - z + 11y),$$

which is in fact

$$P_3(n+1) \supset \langle \alpha^* \rangle (z > 78 \wedge 89 \geq z \wedge n = 89 - z + 11y).$$

We use (J') again, this time with the convergent

$$Q(m): \quad y > 0 \wedge z > 0 \wedge z < 90 \wedge n = 89 - z + 11y \wedge m = \text{floor}((100 - z) / 11) - 1,$$

where  $m = \text{floor}(a / b)$  abbreviates  $(a \geq m \cdot b \wedge (m + 1) \cdot b > a)$ .

It can readily be seen that we can prove in  $P$ :

$$\begin{aligned} P_3(n+1) &\supset \exists m Q(m), \\ Q(m+1) &\supset \langle \alpha \rangle Q(m), \\ \text{and} \quad Q(0) &\supset (z > 78 \wedge 89 \geq z \wedge n = 89 - z + 11y), \end{aligned}$$

which completes the proof. ■

**Appendix C: Example of a Proof of a CFDL-*iff* in  $R$ .**

We sketch a proof of the partial correctness of the factorial program of Section 4.1.

We prove, using standard arithmetical abbreviations

$$\vdash_R [z \leftarrow x; \tau^*(f)] y = x!,$$

where

$$\tau(X): (z=0?; y \leftarrow 1) \cup (z \neq 0?; z \leftarrow z-1; X; z \leftarrow z+1; y \leftarrow y \cdot z).$$

First we prove in  $R$

$$\begin{array}{l} \text{and} \quad (1) \quad [z \leftarrow x] z = x, \\ \quad \quad (2) \quad z = x \supset [\tau^*(f)] y = x!, \end{array}$$

and then, using (H), (G) and (E), we obtain the result.

(1) is trivial using (C). To prove (2) in  $R$  we apply the derived rule (M') as follows: Note that  $\text{var}(\tau) = (y, z)$ . Take

$$\begin{array}{l} \text{and} \quad R: z = x, \\ \quad \quad Q: y = z!, \\ \quad \quad P: z' = z \wedge y' = z!. \end{array}$$

We have left to show

$$\begin{array}{l} \text{and} \quad (3) \quad z = x \supset [(z' = z \wedge y' = z!)(y, z)] y = z!, \\ \quad \quad (4) \quad (z' = z \wedge y' = z!) \supset [(z=0?; y \leftarrow 1) \cup (z \neq 0?; z \leftarrow z-1; (z'=z \wedge y'=z!)(y, z); z \leftarrow z+1; y \leftarrow y \cdot z)] (z=z' \wedge y=z!). \end{array}$$

To prove (3) we use (K), obtaining

$$z = x \supset (\forall y'', z'') ((z'' = z \wedge y'' = z!) \supset y'' = z'')$$

which is an axiom in (B). Proving (4), by (F), (C) and (E), amounts to proving both

$$(5) \quad (z'=z \wedge y'=y) \supset [(z=0?; y+1) \wedge (z'=1 \wedge y'=z?)]$$

which again is an axiom in (B), and

$$(6) \quad (z'=z \wedge y'=y) \supset [(z=0?; z+z-1 \wedge (z'=z \wedge y'=z?))^{(y,z)}] \\ (z'=z+1 \wedge y'(z+1)=z?).$$

The latter we prove by proving in R

$$(7) \quad (z'=z \wedge y'=y) \supset [(z=0?; z+z-1)z \geq 0,$$

and  $(8) \quad z \geq 0 \supset [(z'=z \wedge y'=z?)^{(y,z)}] \wedge (z'=z+1 \wedge y'(z+1)=z?).$

The proof of (7) is quite easy using (C), (D), (E) and (B). For (8) we apply axiom (K) again, to obtain

$$(9) \quad z \geq 0 \supset (\forall y'', z'') ((z''=z \wedge y''=z?) \supset (z'=z''+1 \wedge y'(z''+1)=z?)),$$

which is an axiom in (B). ■

**Appendix D: Example of a Proof of a DL<sup>+</sup>-wff in P<sup>+</sup>.**

Consider the following program

$$\alpha: (x \neq z?; ((x=y?; x \leftarrow x+1) \cup x \leftarrow x+2))^*$$

Assume a state  $J \in N$  for which  $x_J = 0$ . Then, starting from 0,  $x$  gets increased by 2 as long as  $x$  does not "hit"  $z$ . Also, if  $x$  happens to hit  $y$  then one increase by 1 is permitted before the by-2 increases are resumed. Two properties of  $\alpha$  which are of interest in such states and which depend on the values, in these states, of  $z$  and  $y$ , are:

- (a) whether  $x$  can be made to skip  $z$ , and
- (b) whether  $x$  can be made to hit  $z$ ,

and can be written simply as  $loop_\alpha$  and  $\langle \alpha \rangle x=z$  respectively. The behavior of  $\alpha$  in all states of  $N$  in which  $x=0$  depends upon whether or not  $z$  and  $y$  are odd, and also upon whether or not  $y < z$ . The complete situation is given by the following table where  $odd(z)$  and  $even(z)$  stand for  $\exists z'(z=1+2z')$  and its negation respectively:

		$odd(y)$	$even(y)$
$odd(z)$		$loop_\alpha \wedge \neg \langle \alpha \rangle x=z$	$loop_\alpha \wedge \neg \langle \alpha \rangle x=z$ $y > z$
			$loop_\alpha \wedge \langle \alpha \rangle x=z$ $y < z$
$even(z)$		$\neg loop_\alpha$	$\neg loop_\alpha$ $y \geq z$
			$loop_\alpha \wedge \langle \alpha \rangle x=z$ $y < z$

(Note that  $\neg loop_\alpha$  implies that  $\langle \alpha \rangle x=z$ .)

We now prove that

$$x=0 \supset (\text{loop}_e = (\text{odd}(z) \vee (\text{even}(y) \wedge y < z)))$$

is  $N$ -valid by proving the following three formulas in  $P^*$ :

$$(1) (x=0 \wedge \text{even}(z) \wedge (\text{odd}(y) \vee y < z)) \supset \{\text{loop}_e\} \text{true},$$

$$(2) (x=0 \wedge \text{odd}(z)) \supset \{\text{loop}_e\} \text{false},$$

$$(3) (x=0 \wedge \text{even}(y) \wedge y < z) \supset \{\text{loop}_e\} \text{false}.$$

Combining these gives the required result.

(1): We would like to apply the derived rule (T). Denote it by  $P^*$ , and take  $P(n)$  to be

$$(\text{odd}(y) \vee y < z) \wedge z < x \wedge (z - 2x = 0).$$

Certainly we have  $\vdash_{N^*} P(0)$  because  $\vdash_{N^*} (z < 0 \wedge z = 0)$ . Also, to any state  $J \in N$  for which we have

$$J \models (x=0 \wedge \text{even}(z) \wedge (\text{odd}(y) \vee y < z))$$

we also have  $P(n)$  holding, where  $n$  is taken to be  $(1 + (z - x) / 2)$ . We are left then, with having to prove  $\vdash_{N^*} P(n+1) \supset \{\text{loop}_e\} P(n)$  in  $P^*$ , which boils down to proving

$$(P(n+1) \wedge x \neq z) \supset x \neq y,$$

and

$$(P(n+1) \wedge x = z) \supset [x + x = 2]P(n).$$

The first is an axiom and the second can easily be established in  $P$ .

(2): Here we would like to apply derived rule (U) and are looking for a *divergent*  $P$ . We take  $P$  to be simply

$$\text{odd}(z) \wedge \text{even}(x),$$

and it is easy to see that  $(x=0 \wedge \text{odd}(z)) \supset P$  is  $N$ -valid, and hence an axiom of  $P^*$ . Also, one can prove in  $P$  that

$$P \supset (x \neq z \wedge \langle x \leftarrow x+2 \rangle P),$$

so that we have proved  $P \supset \langle \beta \rangle^+ P$  in  $P^+$  and can apply (U') to obtain the result.

(3): Similarly (U') is used, and here the divergent  $P$  is taken to be

$$y < z \wedge \text{even}(y) \wedge ((\text{odd}(z) \wedge \text{even}(x)) \vee (\text{even}(z) \wedge (y < x \neq \text{odd}(x)))).$$

It is easy to see that  $(x=0 \wedge \text{even}(y) \wedge y < z) \supset P$  is  $N$ -valid, and we leave to the reader the task of verifying that  $P \supset \langle \beta \rangle^+ P$  is provable in  $P^+$  (in fact  $P \supset \langle \beta \rangle P$  is), and then an application of (U') completes the proof. ■

**Appendix E: Example of a Proof of a CFDL<sup>+</sup>-uff in R<sup>+</sup>.**

Consider the program

$$\alpha: (u=v? \cup (u \neq v?; u \leftarrow u+2; X; u \leftarrow u-2))^*(f)$$

for which it is the case that

$$\vdash_N (u=0 \supset (odd(v) \equiv loop_\alpha))$$

holds. We sketch the proof in R<sup>+</sup> of one direction, namely that

$$(u=0 \wedge odd(v)) \supset loop_\alpha$$

is N-valid.  $\alpha$  is of the form  $\tau^*(f)$ , and we have by definition

$$\begin{aligned} \sigma: & (y \neq a? \cup (y=a?; x=a?; y \leftarrow b)), \\ \text{and} \quad \tau(X): & ((u \neq v? \cup x \leftarrow b) \cup ((u \neq v? \cup x \leftarrow b); u \leftarrow u+2; (X \cup x \leftarrow b); u \leftarrow u-2))^*(f). \end{aligned}$$

We apply rule (W) taking R to be  $(u=0 \wedge odd(v))$ , Q to be *false*, and  $P(n) \equiv P$  to be

$$(even(u) \wedge odd(v) \wedge u'=u \wedge y'=b \wedge x'=a \wedge x=a \wedge y=a),$$

where  $V=(u,x,y)$ . The third premise of rule (W) is trivially N-valid. Considering the second, we can easily prove

$$(u'=u \wedge y'=b \wedge x'=a \wedge x'=a) \supset \langle y=a?; x=a?; y \leftarrow b \rangle (u=u' \wedge x=x' \wedge y=y')$$

and hence establish, by further propositional reasoning

$$P \supset \langle \sigma \rangle V=V'.$$



Also, one can prove

$$P \supset \langle u \neq v?; u \leftarrow u+2; P^V; u \leftarrow u-2 \rangle V=V',$$

from which the fourth premise follows. We are left with having to prove the first premise. This is done by proving

$$R \supset \forall n \langle x \leftarrow a; y \leftarrow a; P^V \rangle y=b$$

which simplifies to having to prove

$$R \supset \langle x \leftarrow a; y \leftarrow a; P^V \rangle y=b.$$

This again can easily be seen to be provable in **R**, giving the conclusion. ■

## References.

- [1] de Bakker, J.W. Semantics and Termination of Nondeterministic Recursive Programs. In *Automata, Languages and Programming*, Edinburgh. 435-477, 1976.
- [2] de Bakker, 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. Modular Properties of Programs. *Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys.* Vol. 23. No. 3. 1975.
- [7] Banachowski, L., A. Kreczmar, G. Mirkowska, H. Rasiowa and A. Salwicki. An Introduction to Algorithmic Logic; Metamathematical Investigations in the Theory of Programs. In Mazurkiewicz and Pawlak (editors) *Math. Found. of Comp. Sc.* Banach Center Publications. Warsaw. 1977.
- [8] Berman, F. private communication.
- [9] Berman, F. and M. Paterson. Test-Free Propositional Dynamic Logic is Strictly Weaker than PDL. T.R. no. 77-10-02, Dept. of Computer Science, Univ. of Washington, Seattle. Nov. 1977.
- [10] Clarke, E.M. Programming Language Constructs for which it is Impossible to Obtain Good Hoare-like Axiom Systems. Proc. 4th ACM Symp. on Principles of Programming Languages. 10-20. Jan. 1977.

- [11] Constable, R.L. On the Theory of Programming Logics. 9th ACM Symp. on Theory of Computing. Boulder, Colorado, May 1977.
- [12] 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.)
- [13] Dijkstra, E. W. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. Comm. of the ACM. vol 18, no.8. 1975
- [14] Dijkstra, E. W. *A Discipline of Programming*. Prentice-Hall. 1976
- [15] Engeler, E. Algorithmic properties of structures. Math. Sys. Thy. 1, 183-195. 1967.
- [16] Fischer, M.J. and R.L. Ladner. Propositional Modal Logic of programs. Proc. 9th ACM Symp. on Theory of Computing, Boulder, Col., May 1977.
- [17] 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.
- [18] Gabbay, D. Axiomatizations of Logics of Programs. Manuscript, Nov. 1977.
- [19] 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.
- [20] Harel, D. Arithmetical Completeness in Logics of Programs. In *Automata, Languages and Programming*. Springer-Verlag. July 1978.
- [21] Harel, D. On the Correctness of Regular Deterministic Programs; A Unified Survey. Manuscript. Lab. for Comp. Science, MIT. Nov. 1977.
- [22] 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.

- [23] 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.
- [24] Harel, D., A. Pnueli and J. Stavi. *A Complete Axiomatic System for Proving Deductions about Recursive Programs*. Proc. 5th Ann. ACM Symp. on Theory of Computing, Boulder, Col., May 1977.
- [25] Harel, D. and V.R. Pratt. *Nondeterminism in Logics of Programs*. Proc. 5th ACM Symp. on Principles of Programming Languages. Tucson, Ariz. Jan. 1978.
- [26] Hitchcock, P. and D. Park. *Induction Rules and Termination Proofs*. In *Automata, Languages and Programming* (ed. Nivat, M.), IBM North-Holland, 1973.
- [27] Hoare, C.A.R. *An Axiomatic Basis for Computer Programming*. Comm. of the ACM, vol. 12, 576-580, 1969.
- [28] Hoare, C.A.R. *Some Properties of Nondeterministic Computations*. Manuscript. The Queen's Univ., Belfast. 1976.
- [29] Katz, S.M. and Z. Manna. *Logical analysis of programs*. Comm. of the ACM, vol. 19, no. 4, pp. 188-206. Apr. 1976.
- [30] Kleene, S.C. *Introduction to Metamathematics*. D. Van Nostrand. 1952.
- [31] Koenig, D. *Theorie der endlichen und unendlichen Graphen*. Leipzig. 1936. Reprinted by Chelsea, New York. 1950.
- [32] Kreczmar, A. *The set of all tautologies of algorithmic logic is hyperarithmetical*. Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys. Vol. 12, 101-103. 1971.
- [33] Kreczmar, A. *Degree of recursive unsolvability of algorithmic logic*. Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys. Vol. 20, 613-617. 1972.
- [34] Kripke, S. *Semantical considerations on Modal Logic*. Acta Philosophica Fennica, 83-94, 1963.

- [35] Lipton, R.J. A Necessary and Sufficient Condition for the Existence of Hoare Logics. 18th IEEE Symp. on Foundations of Computer Science, Providence, R.I. Oct. 1977.
- [36] Lipton, R.J. and L. Snyder. Completeness and Incompleteness of Hoare-like Axiom Systems. Manuscript. Dept. of Computer Science. Yale University. 1977.
- [37] Manna, Z. The Correctness of Programs. J. of Comp. and System Sciences, vol.3. pp. 119-127. 1969.
- [38] Manna, Z. Second Order Mathematical Theory of Computation. Proc. 2nd ACM Symp. on Theory of Computing, 158-168. 1970.
- [39] Manna, Z. *Mathematical Theory of Computation*. McGraw-Hill. 1974.
- [40] Manna, Z. and J. McCarthy. Properties of programs and partial function logic. In *Machine Intelligence 5*. Edinburgh University Press. 1969.
- [41] Mirkowska, G. On formalized systems of algorithmic logic. Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys. Vol. 22. 421-428. 1974.
- [42] Mirkowska, G. and A. Kreczmar. private communication.
- [43] Meyer, A.R. Equivalence of DL,  $DL^+$  and ADL for Regular Programs with Array Assignments. Manuscript. Lab. for Computer Science. MIT, Cambridge MA. August 1977.
- [44] Meyer, A.R. private communication.
- [45] Morris, J.H. Jr. and B. Wegbreit. Subgoal Induction. Comm. of the ACM. vol. 20. no. 4. April 1977.
- [46] Naur, P. Proof of Algorithms by General Snapshots. BIT vol. 6. 310-316. 1966.
- [47] Owicki, S. A consistent and complete deductive system for the verification of parallel programs. Proc. 8th Ann. ACM Symp. on Theory of Computing, 73-86. Hershey PA. May 1976.

- [48] Parikh, R. A Completeness Result for PDL. To be presented at the Symp. on Math. Found. of Comp. Science, Zakopane, Warsaw, Sept. 1978.
- [49] Parikh, R. Second Order Process Logic. Manuscript. Lab. for Comp. Science, MIT. April 1978.
- [50] Parikh, R. private communication.
- [51] Park, D. Fixpoint Induction and Proofs of Program Properties. In *Machine Intelligence 5*. Edinburgh University Press, 1968.
- [52] Pratt, V.R. Semantical Considerations on Floyd-Hoare Logic. Proc. 17th IEEE Symp. on Foundations of Computer Science, 109-111. Oct. 1978.
- [53] Pratt, V.R. A Practical Decision Method for Propositional Dynamic Logic. Proc. 10th Ann. ACM Symp. on Theory of Computing, San Diego, CA, May 1978.
- [54] Pratt, V.R. private communication.
- [55] Rasiowa, H. On logical structure of programs. *Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys.* Vol. 20. 4319-324. 1972.
- [56] deRoever, W.P. Dijkstra's Predicate Transformer, Nondeterminism, Recursion, and Termination. I.R.I.S.A., Publication Interne No. 37. 1978.
- [57] deRoever, W.P. Equivalence between Dijkstra's predicate transformer semantics and Smyth's powerdomain semantics as found by G. Plotkin. Manuscript, August 1977.
- [58] Rogers, H. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967.
- [59] Salwicki, A. Formalized Algorithmic Languages. *Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys.* Vol. 18. No. 5. 1970.
- [60] Schwarz, J.S. Semantics of Partial Correctness Formations. Ph.D. Dissertation. Syracuse Univ. Dec. 1974.

- [61] Segerberg, K. A Completeness Theorem in the Modal Logic of Programs. Preliminary report. Notices of the AMS, 24, 6, A-552. Oct. 1977.
- [62] Shoenfield, J.R. *Mathematical Logic*. Addison-Wesley. 1967.
- [63] Sokolowski, S. Total Correctness for Procedures. Manuscript. Univ. of Gdansk, Poland. 1977.
- [64] Tarski, A. The semantic conception of truth and the foundations of semantics. *Philos. and Phenom. Res.*, 4, 341-376. 1944.
- [65] Turing, A. Checking a Large Routine. In *Rep. Conf. High Speed Automatic Calculating Machines*. Inst. of Comp. Sci. Univ. of Toronto. Ontario, Can. Jan. 1950.
- [66] Von Neumann, J. *Collected Works*. S. pp. 91-99. Macmillan, New York. 1963.
- [67] Wand, M. A New Incompleteness Result for Hoare's System. Proc. 8th ACM Symp. on Theory of Computing, 87-91. Hershey, Penn. May 1976.
- [68] Wand, M. A Characterization of Weakest Preconditions. *J. of Comp. and System Sciences*, vol. 15. pp. 209-212. 1977.
- [69] Wang, A. An Axiomatic Basis for Proving Total Correctness of Goto Programs. *BIT* vol. 16, 88-102. 1976.
- [70] Winklmann, K. Equivalence of DL and  $DL^+$  for regular programs without array assignments but with DL-formulas in tests. Manuscript, Lab. for Computer Science. MIT, Dec. 1977.
- [71] Winklmann, K. Equivalence of DL and  $DL^+$  for regular programs. Manuscript, Lab. for Computer Science, MIT. March. 1978.

**Biographical Note.**

David Harel was born in London, England on April 12, 1950 and immigrated to Israel with his family in July 1957. He graduated from Netiv-Meir Yeshiva High School in June 1968 and served in the Israeli Defense Forces from Sept. 1968 through Dec. 1971. In July 1974 he received a B.S. in Mathematics and Computer Science from Bar-Ilan University and in August 1976 an M.S. in Computer Science from Tel-Aviv University. He arrived in the U.S., with his wife Varda and their daughters Sarit (now 6) and Nadav (2.5), in Sept. 1976, and received his Ph.D. in Computer Science from MIT in June 1978. He has recently accepted a two year postdoctoral fellowship at the IBM T.J. Watson Research Center at Yorktown Heights, N.Y., starting Sept. 1978.



Errata for MIT/LCS/TR-200, by David Harel.

Page 43. Rule (H) should read:

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

Pages 38-39. Theorem 3.1 and its proof should read:

*Theorem 3.1* (Theorem of Completeness): For any universe  $U$  and  $M$ -extension  $L(M)$  of  $L$ , a  $U$ -sound axiom system  $P(M)$  for  $L(M)$  is  $U$ -complete whenever

- (1)  $P(M)$  is propositionally complete,
- (2)  $L$  is  $U$ -expressive for  $L(M)$ ,
- (3) For any  $k \in K$ , variable  $x$  and  $L(M)$ -wffs  $R$  and  $Q$ ,  
 if  $\vdash_{P(M)} (R \supset Q)$  then  $\vdash_{P(M)} ((M_k)R \supset (M_k)Q)$ ,  
 if  $\vdash_{P(M)} (R \supset Q)$  then  $\vdash_{P(M)} (\exists x R \supset \exists x Q)$ , and
- (4) For any  $k \in K$  and  $L$ -wffs  $R$  and  $Q$ ,  
 if  $\vDash_U R$  then  $\vdash_{P(M)} R$ ,  
 if  $\vDash_U (R \supset (M_k)Q)$  then  $\vdash_{P(M)} (R \supset (M_k)Q)$ , and  
 if  $\vDash_U (R \supset \neg(M_k)Q)$  then  $\vdash_{P(M)} (R \supset \neg(M_k)Q)$ .

*Proof:* We have to prove that if  $P$  is an  $L(M)$ -wff such that  $\vDash_U P$ , then  $\vdash_{P(M)} P$ . By the propositional completeness of  $P(M)$  we can assume that  $P$  is given in conjunctive normal form, and we proceed by induction on the sum  $n$ , of the number of appearances of  $M$  and the number of quantifiers prefixed to non first-order formula, occurring in  $P$ . In the case  $n=0$ ,  $P$  is first-order and by the first line in assumption (4) it is provable if it is  $U$ -valid. Assume that  $n>0$  and that the theorem holds for any formula with  $n-1$  or less appearances of  $M$  and such quantifiers. If  $P$  is of the form  $P_1 \wedge P_2$  then we have  $\vDash_U P_1$  and  $\vDash_U P_2$ , both of which have to be proved in  $P(M)$ , so that we can restrict our attention to a single disjunction. Without loss of generality we can, therefore, assume that  $P$  is of one of the forms:

$$P_1 \vee (M_k)P_2, \quad P_1 \vee \neg(M_k)P_2, \quad P_1 \vee \exists x P_2 \quad \text{or} \quad P_1 \vee \neg \exists x P_2,$$

where  $k \in K$ , and the right-hand side disjunct is *not* first-order. Thus we are guaranteed that in each case  $P_2$  has less than  $n$  appearances of  $M$  and such quantifiers. Let us use  $\rho$  to denote  $(M_k)$ ,  $\neg(M_k)$ ,  $\exists x$  or  $\neg \exists x$  according to which is the case.

$L$  is expressive for  $L(M)$ , and so for any  $L(M)$ -wff  $Q$  there is some  $L$ -wff  $Q_L$  which is equivalent to  $Q$ . We have then  $\vdash_U (\neg P1_L \supset \rho P2_L)$ . Now, using assumption (4) (since  $P1_L$  and  $P2_L$  are  $L$ -wffs) we also have

$$(*) \quad \vdash_{P(M)} (\neg P1_L \supset \rho P2_L).$$

Now surely, by the definition of  $P1_L$  and  $P2_L$ , we have  $\vdash_U (\neg P1 \supset \neg P1_L)$  and  $\vdash_U (P2_L \equiv P2)$ . Both these last formulae have less than  $n$  appearances of  $M$  and such quantifiers, and hence by the inductive hypothesis

$$(**) \quad \vdash_{P(M)} (\neg P1 \supset \neg P1_L) \quad \text{and}$$

$$\vdash_{P(M)} (P2_L \equiv P2).$$

By assumption (3) (together with the propositional completeness when the second and fourth of the above cases are considered) we can obtain from the latter

$$(***) \quad \vdash_{P(M)} (\rho P2_L \supset \rho P2).$$

From (\*), (\*\*) and (\*\*\*) we get, using propositional reasoning,  $\vdash_{P(M)} (\neg P1 \supset \rho P2)$ , or  $\vdash_{P(M)} (P1 \vee \rho P2)$ .  $\square$