MicroD

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

**PROPOSITIONAL DYNAMIC LOGIC OF REGULAR PROGRAMS
WITH TESTS, CONVERSE AND INFINITE COMPUTATIONS**

by

FRANCIS J. SMITH, M.A.

A thesis submitted to
the faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of

Master of Science
in Information and Systems Science
Department of Mathematics and Statistics

Carleton University
Ottawa, Ontario
28 March 1988

© copyright
1988, Francis J. Smith

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-40627-5

The undersigned recommend to the faculty of Graduate Studies
and Research acceptance of the thesis
**PROPOSITIONAL DYNAMIC LOGIC OF REGULAR PROGRAMS
WITH TESTS, CONVERSE AND INFINITE COMPUTATIONS**
submitted by Francis J. Smith, M.A.
in partial fulfilment of the requirements for
the degree of Master of Science

Thesis Supervisor

Chairman, Department of Mathematics and Statistics

Carleton University

1988

To Alyssa, Evelyn and (especially to) Monika

with love

ABSTRACT

This thesis comprises a study of the propositional dynamic logic of a matrix of programming schemes. The compositional language of regular or iterative programs is the basis of the study. This language is then complemented by tests, converse and the ability to make assertions about infinite computations. Axiomatisations, starting from the well established Segerberg axioms, are provided. Our focus is on the "denotational meaning" of well-formed program expressions and on the validity of expressions about them with respect to standard, non-standard and syntactically motivated classes of Kripke models. Other semantical issues dealt with include notions of consequence and compactness, model size properties, soundness, the complexity of deciding whether a formula is valid or not, and (especially) completeness. Each of these issues is taken up with respect to the different distinguished classes of models. The thesis ends with a comparison of the expressive capability of each propositional dynamic logic within the matrix.

PREFACE

Dynamic logic is a logic for reasoning about change-of-state, and is most often regarded as a logic for reasoning about computer programs. Because it represents the marriage of two well established disciplines - pure modal logic with the calculus of relations - the invention of dynamic logic, in the mid-1970's by Vaughan Pratt [PRA76], represents a significant advance in the field of program logics and program semantics. In this thesis we make a comprehensive study of the important developments in the propositional manifestation of dynamic logic from its inception, in M. Fischer and R. Ladner's ground breaking article [F&L77], to the present.

outline

Chapter one serves as a basic introduction to the thesis. It provides the reader with a basic understanding, motivation, need, purpose and requirements of any logic of programs and of dynamic logic in particular. Dynamic logic itself is introduced, both syntactically and semantically as an extension of modal logic. The idea that programs can be interpreted as relations between the states of a computer is accepted.

Chapter two presents the bulk of logical syntax that will be used throughout. Various nondeterministic propositional dynamic logics are introduced. The most elementary of our logics is test free propositional dynamic logic of regular or iterative programs. All other logics considered are extensions to this elementary version of dynamic logic. We consider conditional branching, converse execution and the

capability to make assertions about infinite computations.

Chapter three gives the basic set theoretic semantics for the logics that were introduced in chapter two. Several different classes of models based on the well-known Kripke structures of modal logic are distinguished. With semantics in place, the soundness of these logics is easily established. (Thus, pending the existence of models, all the logics considered are consistent.) It is demonstrated that even the most elementary of our logics fails to be semantically compact. A useful property for demonstrating completeness and decidability of some propositional dynamic logics is the "compact model property". It is shown that most of the logics considered in this thesis respect this or the less restrictive, but related, finite model property. At the end of this chapter the complexity of deciding the validity of formulae is briefly considered.

The objective of chapter four is to demonstrate completeness for the various logics under our consideration with respect to the semantics developed in the previous chapter. The methods are those of classical propositional modal logic - categorical models and filtrations - suitably modified for application to propositional dynamic logic. This is the most difficult part of the thesis and should be considered its prime focal point.

Finally, in chapter five, the expressive power of each of the logics is compared. It is shown that, at every stage, each extension adds to the expressive power of propositional dynamic logic.

Background

We have endeavored to make this thesis stand by itself. However, some (small) assumptions about the reader's background have been made. Throughout, a rudimentary familiarity with the classical propositional logic is assumed. Those

Preface

wishing to remind themselves of these fundamentals are referred to [B&S69; chap.2] or [C&K73;chap.1]. Although it is not essential, some knowledge of propositional modal logic and Kripke frames would certainly benefit the reader. A good introduction to propositional modal logic with Kripke semantics is [CHE80]. (A somewhat more advanced introduction is [LEM77] - see also [BOO79] and [SEG71].) In the introduction the reader is asked to (temporarily) assume some formulation of the classical predicate calculus such as [B&S69;chap.3] or [C&K73;chap.1]. Chapter three, section 3.5 briefly deals with the problem of deciding the validity of a formula. However, due to the brevity of our treatment of the complexity of solutions to this problem, the reader with some knowledge in this area would have a definite advantage. Those wishing information on complexity theory may consult [AHU74]. To a lesser extent, the same may be said of a background in formal language theory when it comes to understanding parts of the introduction and portions of chapter five. (In the latter, explicit use is made of the well known uvxyz-theorem of context-free languages.) For background information on formal language theory consult either [H&U69] or [GIN66].

Acknowledgements

Financial and moral support for this thesis has come from my employer, Telesat Canada and its staff. In particular, I would like to thank the staff of Telesat's library (especially Eileen Foster and Margaret Colwin) who obtained several articles of the bibliography that proved difficult to find. (There are very few copies of [SEG71] in Canada - and none in Ottawa!) This thesis was written on TEXT/1000, a word processor written by Frans Kes for in-house use by the Satellite Control Division of Telesat Canada. For the layout and professional "look" of this thesis Frans deserves much of the credit.

The work that follows has benefitted from the input a several people, but three —deserve special mention. At VERY short notice the "team" of Monika Plettenberg and

Preface

Rita Smith were enlisted for proof reading. My supervisor, Dr. John Poland, has read the whole thesis and his input has been very helpful. For this (and their patience) I wish to thank them. Ultimately, most thanks go to my family - Monika, Alyssa and Evelyn - for allowing me to do this. They deserve far better, and they certainly deserved it faster!

TABLE OF CONTENTS

	<u>page</u>
ABSTRACT	iv
PREFACE	v
TABLE OF CONTENTS	ix
LIST OF FIGURES	xi
1. INTRODUCTION:	1
1.1 Prolegomenon	2
1.2 Logics of Programs	5
1.3 The Meaning of Programs	9
1.4 Possible World Semantics for Programs	12
1.5 Modal Logic - the Dynamic Logic of a Distinguished Program	13
1.6 The Compositional Structure of Computer Programs	15
1.7 Applications of Dynamic Logic	21
1.8 Determinism Versus Nondeterminism	22
Chapter 1 Notes	25
2. LOGICAL PRELIMINARIES	38
2.1 The Language of Elementary Propositional Dynamic Logic	38
2.2 Extensions to the Language of Elementary Propositional Dynamic Logic	42
2.3 Axioms and Rules	43
2.4 Immediate Results: Normal Propositional Modal Logics	50

Table of Contents

2.5 Immediate Results: Propositional Dynamic Logics	52
2.6 Maximality and Lindenbaum's Lemma	59
Chapter 2 Notes	62
3. SEMANTICS	63
3.1 Standard and Non-Standard Kripke Models	63
3.2 Soundness and Consistency	71
3.3 Semantic Compactness and Semantic Consequence	76
3.4 Collapsed and Finite Model Properties	80
3.5 Decidability and Complexity	88
Chapter 3 Notes	94
4. COMPLETENESS	98
4.1 Canonical Kripke Models	99
4.2 Fischer-Ladner Filtrations	104
4.3 The Completeness With Respect to Standard Kripke Models	106
4.4 The Kozen-Parikh Model	111
Chapter 4 Notes	115
5. COMPARING EXPRESSIVE POWER	117
5.1 The Expressive Power of Tests	118
5.2 The Expressive Power of Reverse Computations	126
5.3 The Expressive Power of Infinite Computations	127
5.4 Summary of Results on Expressive Power	129
Chapter 5 Notes	130
6. BIBLIOGRAPHY	131

LIST OF FIGURES

	<u>page</u>
5-1 Berman-Paterson model	119
5-2 Layered Berman-Paterson model	125

CHAPTER ONE

INTRODUCTION

Dynamic logic is most often associated with programming because it was invented and first studied by computer scientists for reasoning about programs. It was only in afterthought that the epithet "dynamic" was applied, in recognition of the logics potential for reasoning about any of the "agents" which can bring about change. Dynamic logic, in fact, is a formal logic for reasoning about *generic exercises of agency*. (NOTE: Thompson) It is because programs bring about a change-of-state (i.e., an "exercise") of computers that dynamic logic is appropriate for reasoning about programs. The most notable application of dynamic logic in a non-programming milieu is as a system for reasoning about actions in general. In this thesis, however, dynamic logic will be treated as a logic of programs. (NOTE: name)

Our attention will be restricted to the sentential aspect of dynamic logic. The relationship between first-order dynamic logic and propositional dynamic logic is that of the relationship between the first-order predicate calculus and classical propositional logic. As a form of reasoning about actions or programs it can be as limited as classical propositional logic is in formalizing logical arguments. On the other hand, "... a thorough understanding of [propositional dynamic logic] is a prerequisite to obtaining a good grasp of the more complicated, albeit more applicable, systems, just as classical propositional logic is fundamental to the understanding of first-order predicate calculus." [F&L;pp.210] Propositional dynamic logic has great theoretical interest of its own and, in fact, it has been suggested that it has more theoretical interest than first-order dynamic logic. (NOTE: theoretical interest)

The variety of programs considered in this thesis are the nondeterministic iterative

programs (sometimes called regular programs because of their association with Kleene's regular expressions). This class is extended to consider conditional tests, backward execution and infinite execution because these are, in many ways, simple extensions that add real expressive power to the class of nondeterministic iterative programs. On the other hand, recursion or variable assignment are not considered. Even though these program types are deservedly regarded as fundamental concepts in computer science, their inclusion cannot be regarded as simple. Variable assignment is a first-order concept that is under the jurisdiction of first-order dynamic logic and recursion requires us to consider the richness of context-free languages which is beyond the scope of this thesis. Because of its predominance, a contraction of our class of programs to those which are strictly deterministic, is briefly considered.

In the next section there is a discussion of why and how we should formulate a logic for reasoning about programs. The remainder of the introduction will then be dedicated to showing that dynamic logic satisfies these conditions - invariably doing better than most other logics of programs. Along the way, the reader is introduced to the syntax and semantics of propositional dynamic logic. This approach shows how dynamic logic may be regarded, both in its syntax and its semantics, as a non-trivial extension of modal logic with semantics based on relational frames. Dynamic logic is then favourably compared with its famous predecessor, the Floyd-Hoare logic.

1.1 Prolegomenon

Why formalize reasoning about programs? Indeed, why bother to formalize reasoning of any sort at all? From a practical point of view it is usually assumed that any type of reasoning, including the type that has programs as its objects, can be better transacted with the aid of formal logic. The sort of reasoning that goes on at the first-order predicate level, however, will seem rather straightforward when compared to the complexity of even simple computations. Because computer programs can become very complicated, we can, comparatively speaking, benefit greatly by

formalizing a logic about them. Thus, the [formal] ". . . study of programming, in particular of program verification, in the context of programming logics will lead to a more coherent view of the subject. It will also suggest, by analogy with mathematical logic, a rich line of theoretical inquiry which will eventually provide a much deeper basis for the design of useful program verification systems." [CON77;pp.270] {NOTE: theoretical inquiries}

The most famous application of programming logics is to the expression and derivation of the correctness of a given program (otherwise known as *program verification*). (A program is said to be correct when it actually does what it is supposed to do - more on this below.) There are many other applications for programming logics, perhaps more pragmatic than theoretically interesting. For example, it may be useful to know whether two programs are, in some given sense, equivalent. Thus, by formal reasoning about a complex program we may more easily simplify it. Moreover, because formalization usually lends itself to mechanization we should be able to construct a program verifier which would check that a given program satisfies a given condition. (NOTE: verifier)

Given that there are good motives for formalizing reasoning about actions or programs, what requirements should the resulting logics demonstrate? Van Emde Boas [EB78; pp.9] has suggested four criteria.

The foremost of Van Emde Boas' requirements of a logic for reasoning about programs is that it should be able to express the major properties of programs (such as correctness criteria, equivalence, etc.). Also, although Van Emde Boas does not indicate, the expressions for program objects of the logic must include a major portion of the building blocks used by programmers to create programs (e.g., variable and array assignment, recursion, looping and iteration, branching, if-then-else and case constructs, etc.). If our logic fails to be expressive in these ways it can hardly be of any use. In this introduction it will be shown how even the sentential version of dynamic logic fulfills a large part of this requirement. In both of these notions of expressive power: the ability to express qualities of programs, and in the ease in which

complex programs may be expressed, dynamic logic has become the "yardstick" of comparison for other logics of programs.

The logic should also exhibit an intuitive metalogic, in the sense that obvious truths and valid arguments should follow with "reasonable" ease. This constraint, coupled with the first requirement, says that any logic for reasoning about programs must be able to make *correct assertions about programs*. In a nutshell, the purpose of this thesis is to show that, indeed, sentential level dynamic logic demonstrates this quality.

Easy mechanization is a desirable (albeit nonessential) quality that a logic for reasoning about programs should exhibit. As we have already mentioned, formalization lends itself to mechanization. Essential to the property of practical mechanization is an efficient method for deciding whether a formula is a theorem or whether that formula is "true". (NOTE: decidability) Of course, questions of decidability are also of considerable theoretic interest. This is especially true when it comes to comparing different logics of programs. Fortunately, dynamic logic compares well with its competitors in this regard. In fact, the problem of deciding whether a given formula of propositional dynamic logic is valid, is nearly always of single exponential complexity - the same as the much simpler classical propositional logic (see chapter three, section 3.5).

The last, but nevertheless not least, of Van Emde Boas criteria concerns the degree of syntactic and semantic compositionality of the logic. Roughly, this condition (attributed to Frege) stipulates that the shape (syntax) and meaning (semantics) of an expression should be composed of the shape and meaning of its constituent parts. Since Frege, this principle has been central to the development of modern mathematical logic and will also be so for this thesis.

1.2 Logics of Programs

During their execution, computer programs can change the value of variables and, in so doing, they often alter the truth status of propositions about them. Thus, the truth status of a proposition about a program is generally *dynamic*. As a simple example, let *FACTORIAL* be the program which, starting with its internal variable x being assigned an integer n read as input, calculates the value of $n! = n(n-1)(n-2) \dots (2)(1)$. Assume the following primitives: **input**; variable assignment (denoted $\leftarrow$); simple logical and arithmetical expressions (where $\cdot$ and $-$ stand for multiplication and subtraction, respectively); the familiar *while* programming structure; and **output**. *FACTORIAL* may look something like:

```

program FACTORIAL
  input x
  a ← 1
  while not x=0 do
    a ← x·a
    x ← x-1
  enddo
  output x
end.

```

During the execution of *FACTORIAL*, the proposition $x=0$ is false. However, if *FACTORIAL* terminates $x=0$ is true. This capability to alter a proposition's truth status is an essential aspect of a program (or, for that matter, any exercise of agency). It is this *dynamic* aspect which a logic for reasoning about programs or actions must capture.

In general, a program takes an input, perhaps a sequence of numbers $n_1, n_2, \dots, n_i$ and according to some transformations yields an output vector, say $m_1, m_2, \dots, m_j$. (Of course, there are always initial conditions on the type of input, just as there are terminating requirements on the type of output for our program.) Our confidence in the program is reflected by the following assertion:

*After executing the program (with input satisfying the initial conditions)
the output will satisfy the terminal conditions.*

(Reasoning about the consequences of actions in general, is very similar. For example, consider the statement "after painting a box green, the box is green". (NOTE: Pratt's example #1) This sentence has the form "after [subject(s)] performing action A it will be the case that p ", where p is a statement expressing some state-of-affairs, analogous to the terminal conditions on the output of a program.)

Let $\mathcal{L}_{CPL}$ be the language of classical propositional logic or CPL. We take $\mathcal{L}_{CPL}$ to be composed of a countable set of propositional letters denoting atomic or primitive formulae, the constant symbol $\perp$ denoting falsum, the binary alteration operator $\vee$ and the unary negation operator $\neg$. The set of well-formed formulae of CPL is the smallest set Σ_{CPL} containing all atomic formulae, $\perp$ and compound formulae of the form $p \vee q$ and $\neg p$ where $p, q \in \Sigma_{CPL}$. The language of classical first-order (predicate) logic, or CQL, is denoted $\mathcal{L}_{CQL}$. $\mathcal{L}_{CQL}$ is defined as $\mathcal{L}_{CPL}$ above by substituting the set of propositional letters with a set of variable letters, a set of function letters and a set of predicate letters, and by adding quantifier symbols. Some standard definition of well-formed formulae of CQL (e.g., [B&S69; chap.3] or [C&K73; chap.1]), denoted Σ_{CQL} , is assumed. Also, let the abbreviating constant $\top$ (denoting verum) and binary operators of conjunction (denoted $\&$), material implication (denoted $\rightarrow$) and material equivalence (denoted $\equiv$) be defined in the usual way (see chapter two, section 2.1 for details).

In this section, a means of extending the language of classical logic to enable us to formalize our reasoning about programs is investigated. Let us suppose that we know what programs are and how to build them. Let Π be the set of well-formed program expressions written in an assumed programming language. (In section 1.6 an abstract programming language that builds compound program expressions from basic or primitive ones is introduced.)

The first principle contribution to the discipline of programming logics is the so-called Floyd-Hoare logic. This logic is an extension to first-order predicate logic. Let the symbols p and q stand for some first-order assertions made in $\mathcal{L}_{CQL}$ about program $a \in \Pi$. The partial correctness assertion, or PCA, of a with respect to

precondition p and postcondition q is denoted by

$$p\{a\}q$$

and formalizes the assertion

if p is true before the execution of the program a , then q will be true when a terminates.

(NOTE: total correctness assertions) Σ_{CQL} is thus extended to Σ_{FHL} , the set of well-formed formulae of Floyd-Hoare logic, by including all PCA's of any $a \in \Pi$. Note that PCA's in Floyd-Hoare logic have special status and cannot be combined with other operators of the language to form new expression (e.g., $\neg(p\{a\}q)$ is not a well-formed formula of Floyd-Hoare system). This artificial restriction turns out to be a major difficulty with Floyd-Hoare logic that is nicely avoided by dynamic logic. (NOTE: Floyd-Hoare history)

E.W. Dijkstra's notion of the weakest liberal precondition of program a with respect to postcondition q (written $wlp(a, q)$) specifies the conditions that guarantee the truth of q given that a terminates. In Floyd-Hoare logic $wlp(a, q)$ is expressed as the PCA formula $\top\{a\}q$. The condition that characterises that the program a will terminate satisfying q is called the weakest precondition, denoted $wp(a, q)$. Unfortunately, since PCA's cannot themselves express pure termination, $wp(a, q)$ is not expressible in Floyd-Hoare logic.

A notion similar to $wlp(a, q)$ is R.W. Floyd's strongest postcondition of a program a with respect to precondition p , written $sp(a, p)$. $sp(a, p)$ is a formula which is only true when a terminates after starting in a condition satisfying p . The $wlp(a, q)$ and $sp(a, p)$ formulae are related - in fact they provide about the same amount of information but they do so from the perspective of different directions of execution. In Floyd-Hoare logic there is a duality that exists between the forward and backward execution of a program. Let a^c denote the converse of the program a , then this duality principle is

expressed by the following equivalence:

$$p(a^c)q \equiv \neg q(a) \neg p.$$

Thus, $sp(a,p)$ is the PCA $\perp \{a^c\} \neg p$.

The following is the approach of dynamic logic. Extend Σ_{CPL} (and Σ_{CQL}) to Σ_{PKL} (and Σ_{QKL}) by adding the rule: if p is a well-formed formula and $a \in \Pi$ then $\langle a \rangle p$ and $[a]p$ are also well-formed formulae. The notation $\langle a \rangle p$ denotes the statement

when program a finishes proposition p may be true

and let $[a]p$ denote the statement

when program a finishes proposition p will always be true.

Notice that these statements are dual:

$$\langle a \rangle p \equiv_{\text{def}} \neg [a] \neg p.$$

In the example at the beginning of this section, $x=0$ may be false, but $\langle \text{FACTORIAL} \rangle x=0$ is certainly true (i.e., there is at least one initial value of x that ensures the termination of *FACTORIAL*). Also, the formula $[\text{FACTORIAL}] x=0$ is certainly false (i.e., not all initial values of x will cause *FACTORIAL* to terminate). The reader familiar with modal logic will notice a similarity. The notations $\langle a \rangle$ and $[a]$ (called modalities) correspond to the dual unary connectives $\diamond$ and $\Box$ of modal logic respectively. Also, when restricted to the domain of a single action or program, the statement "necessarily p " may be intuitively interpreted as " p will be true in all cases where the program terminates". (The extension given above actually defines the well-formed formulae of normal multi-modal logic or just KPL for short. In section 1.6 we define the well-formed expressions of both programs and formulae of propositional dynamic logic. For the reader who is not familiar with modal logic, section 1.5 should help to clarify

this connection.)

It is a simple matter to reduce PCA's to the notation of dynamic logic. The PCA, $p(a)q$ abbreviates the formula $p \rightarrow [a]q$ of dynamic logic, i.e.,

$$p(a)q =_{\text{def}} p \rightarrow [a]q .$$

Also, the weakest liberal precondition of a with respect to q is easily defined:

$$wlp(a,q) =_{\text{def}} \neg \{a\}q =_{\text{def}} [a]q .$$

Thus, were it permissible, $([a]q)\{a\}q$ would express the fact that no weaker (liberal) precondition than $[a]q$ would permit the postcondition q of a . Likewise:

$$sp(a,p) =_{\text{def}} \neg \{a^c\} \neg p =_{\text{def}} \langle a^c \rangle p .$$

Thus, were it permissible, the formula $p(a)(\langle a^c \rangle p)$ would express the fact that no stronger postcondition than $\langle a^c \rangle p$ would follow from precondition p . Also, because it is so lop-sided in its approach to program termination, Floyd-Hoare logic is strictly less expressive than dynamic logic. The termination of a program a is given by the formula $\langle a \rangle \top$ (i.e., " a can terminate"). Termination coupled with the fact that whenever a terminates it satisfies postcondition q (i.e., $wlp(a,q)$) gives

$$wp(a,q) =_{\text{def}} \langle a \rangle \top \& [a]q .$$

1.3 The Meaning of Programs

Ostensibly this thesis is about a propositional level logic for reasoning about programs. But what are programs? In this section, the reader is introduced to the different approaches to program semantics. Semantic descriptions of programs fall into

three overlapping categories.

The operational semantics of a program describes how it solves its given task. According to this view, the meaning of a program is given by a translation into a sequence of unambiguous, primitive and clearly understood set of instructions of some abstract machine. (Thus, it should be noted, this view is not completely satisfactory until a semantics is supplied for the rudimentary base instruction set of the abstract machine.) In practice, the "meaning" of a program on any given "real" machine is specified by the language compiler. (NOTE: operational semantics) This approach is analogous to knowing the whereabouts of something by being given directions on how to get there. Although the latter does contribute to our knowledge of a specific location, and may indeed enable us to derive the absolute location, it is nevertheless still only relative knowledge. Finally we note that the operational approach is strictly non-compositional thereby opposing one of the tenets of the proceeding prolegomenon. This thesis will not be concerned with this approach.

The axiomatic approach is to describe programs by making assertions about them - actually fixing the meaning of a program by formally specifying what is true of it. The approach here is to provide conditions for manipulating programming language constructs. Assertions made of a program are from a set of compositionally defined well-formed expressions and their "truth" depends on their being derivable from the rules and axioms of some chosen logic, say the Floyd-Hoare logic. These conditions are supplied as a set of rules, axioms and derivable theorems in the chosen logic. This technique allows us to say what is "true" of a program (and, if the logic is "complete", then it allows us to say all of what is "true" of it), but there are no immediate guarantees that we are able to derive its "real" declarative meaning. (NOTE: success in axiomatic approach) Because it attempts to fix meaning through indirect knowledge, the axiomatic approach is similar to the operational approach. The method is analogous to diagnosing the cause of a disease through the symptoms it displays. Just as symptoms do not always lead to successful diagnosis, the rules and axioms of the logic for manipulating programming constructs may not totally encapsulate the meaning of the definable programs. This is the *adequacy problem*, and is one that all logics of

programs must address.(NOTE: adequacy of Floyd-Hoare logic)

The final approach, called declarative or denotational semantics is based on model theoretic (i.e., mathematical) semantics - and is therefore essentially compositional. Under this view, each element of the syntax of a programming language is given a specific valuation, perhaps a number, a truth value or a function. The meaning of compound programming constructs, such as user defined data types and subroutines, are implied inductively from the valuation of the basic elements of the syntax. (Considering the analogy of the last paragraph, this approach provides the "absolute" meaning of a program.) For us, the declarative semantics is the only acceptable approach to fixing the meaning of a program.(NOTE: Scott-Strachey denotational semantics)

Like its predecessor, Floyd-Hoare logic, dynamic logic is essentially an attempt to "diagnose" the meaning of a program from the "symptoms" it displays. In short, it is one of many axiomatic approaches to program semantics and, as such, it does not represent anything very novel. However, there is a semantic aspect to dynamic logic itself that demands that each programming construct be given a declarative meaning. Therefore, in spirit, dynamic logic also embraces the denotational approach. The axiomatic side of the logic gives legal manipulations of programming constructs and the semantical side of the logic captures their "real" (that is, mathematical) denotation. Indeed, part of the appeal of dynamic logic is in the purely syntactical characterization of the "semantical" notions of partial correctness assertions, weakest (liberal) preconditions, and strongest postconditions of programs. The distinction between pure syntax and pure semantics also helps us resolve the adequacy problem for dynamic logic: adequacy reduces to the problem of completeness (and for this problem we are now allowed to borrow from the successful techniques of modal logic - see chapter four).

Although these are not mutually exclusive approaches each has its own separate appeal: "It may appear that the operational method is likely to be of most value to the implementer [vis a vis compiler writing], the axiomatic to the programmer [vis a vis

concern for a programs correctness], and the denotational to the language designer [vis a vis understanding the absolute meaning of a programming construct]. This impression is not so far from the truth, but it is something of an oversimplification. Neither the three methods nor the three sorts of activity are as disjoint as the division suggests, and there are techniques for moving from one formulation to another in studying a language." [STR77;pp.24] Despite the proviso that ends this quote it is hoped that dynamic logic will appeal directly to the programmer and language designer.

1.4 Possible World Semantics for Programs

How should we interpret the program modality $[a]$? Again, modal logic offers a clear possibility. For Kripke [KR163], the interpretation of the extra connective $\Box$ of classical modal logic is by a structure (W, π) called a *Kripke frame*. Inspired by the famous Leibnizian fantasy, Kripke called the non-empty set W the set of all *possible worlds* (NOTES: Leibniz, ontology of possible worlds) If a proposition correctly described a possible world then it is said to be true in that world, otherwise it is false. Thus, Kripke believed, the concept of possible worlds was critical for an understanding of the primitive modal notions of *necessity* and *possibility*: what is true of one world (i.e., *possibly true*) may not be true in all worlds (i.e., *necessarily true*). Also, π is a binary relation on W called the *accessibility relation*. The accessibility relation specifies exactly what worlds are possible (i.e., accessible), relative to others. For example, if u and v are possible worlds (i.e., $u, v \in W$) then $(u, v) \in \pi$ means that v is possible relative to (or equivalently, accessible from) u . For a proposition p to be necessarily true (denoted $\Box p$) at some possible world w , p must be true at all possible worlds accessible from w . For a proposition to be possibly true (denoted $\Diamond p$) at w , there must be at least one possible world which is accessible from w and at which p is true. Can possible world semantics be adopted for programs? If we are only interested in the before-after reasoning about program (i.e., programs as "black boxes") then the answer is yes - with little work at all.

A computer program, after a process of translation to the machine level, is a series of instructions "understood" by the computer. These instructions cause the machine to alter its internal state (i.e., to go from the prior state into another). A final state is arrived at when the program terminates. On an abstract level this view suggests that the complete description of a computer program can be regarded as a set of sequences of accessible states (abstracted, say, from the internal states of the computer). Each sequence specifies a possible execution of the program. The first and last elements of each sequence gives the *initial* and *final* state of the execution. We can trace the "flow" of a program via these sequences, going from one state to another, until we reach a final state. A program thus contributes (or perhaps we should say, *stipulates*) both the accessible states and the flow between these states. (NOTE: transworld identity) If we are only interested in the net behaviour of a program then we may regard it as a relation between its possible initial and final states. This relation is sometimes referred to as the *input/output relation* and the members of this relation are often called *transitions*. The correspondence between this view of programs and Kripke frames is immediately apparent: Possible program states are to possible worlds as the input/output relation is to world accessibility. (NOTE: process logic) Also notice that the input/output relation supports nondeterminism - it is possible that several different transitions of a program have the same final state. An input/output relation that is a partial function, however, is in this sense deterministic.

(The net effect of actions in general is in how it changed the state of "affairs of the environment". Therefore, general actions correspond to Kripke frames in the same way as programs do: states of "affairs of the environment" are to possible worlds as the transition between these states is to world accessibility.)

1.5 Modal Logic - the Dynamic Logic of a Distinguished Program

Dynamic logic is often referred to as a multi-modal logic because of its legion of irreducible modalities. When restricted to the domain of a single distinguished

program a , dynamic logic becomes ordinary modal logic: Let $\langle a \rangle$ and its dual $[a]$ be denoted by the modal connectives $\Diamond$ and $\Box$ respectively. (NOTE: modal logic) Now, the statement "necessarily p " or $\Box p$ may be intuitively interpreted as

p (will be true) in all cases where a terminates.

Likewise, the symbol $\Diamond$ means

p in at least one case where a terminates

The language of propositional classical modal logic, $\mathcal{L}_{PML}$, is just $\mathcal{L}_{CPL}$ supplemented by $\Diamond$. The unary connective $\Box$ is just the dual of $\Diamond$ i.e., $\Box p = \neg \Diamond \neg p$. The set of well-formed formulae is extended by simply allowing $\Diamond p$ where p is any well-formed formula. Denote this expanded set of well-formed formulae Σ_{PML} .

In the last section we promoted the idea of a program (up to the detail of input/output) as a relation between states. Given the Kripke frame (W, π) that gives the before/after "meaning" of the distinguished program a , can we determine the corresponding modal logic for reasoning about a ? Conversely, given a set of axioms and/or rules that supplement those of classical logic and equip us to reason about the input/output behaviour of program a , can we determine the corresponding Kripke frame - i.e., the "meaning" of a ? These questions are intimately related to completeness and soundness questions of modal logic i.e., given a modal logic, under what Kripke frames are its theorems valid and the valid formulae theorems? (NOTE: correspondence theory)

Consider, for example, the sound formula schema

$$(Ax.det) \quad \Diamond p \rightarrow \Box p$$

$Ax.det$ is valid in any Kripke frame (W, π) where π is a partial function i.e., if $(u, v_1) \in \pi$ and $(u, v_2) \in \pi$ then $v_1 = v_2$. Thus, $Ax.det$ is axiomatic when a is a deterministic program

(see section 1.8) - a program that when started in a particular state will be guaranteed to terminate in *at most* one state. We now show this within our pre-formal framework: If $\Box p$ is "valid" (meaning "always-true") then α , when started in any state, can terminate in a state where p is "true". That is, for all $u \in W$, there is a state $v \in W$ such that $(u, v) \in \pi$ and " p is true at state v ". When coupled with the fact that π is a partial function, this implies that, for all states $u \in W$, there is exactly one state $v \in W$ such that $(u, v) \in \pi$ and " p is true at state v ". Therefore, if $\Box p$ is "valid" then, for any $(u, v) \in \pi$, " p is true at state v ", i.e., $\Box p$ is "valid".

1.6 The Compositional Structure of Computer Programs

In this thesis an abstract syntax for computer programs is considered. In many ways the expressive power of this language is very restrictive. General variable and array assignments must be considered primitive. Thus, it is the pure logical structure of program schemes that is studied. Recursion or calls to subroutines are usually thought of as fundamental programming language constructs, but these too are not considered. (We will have a little more on these omissions at the end of this sections.) On the other hand, the programming language that is defined is suitable for analyzing a great many aspects of programming languages (including an esoteric backtracking capability and a very rich capability for expressing controlled branching which is particular to a very few AI languages). The language also considers nondeterminism as a fundamental property of programs just as it is considered in the theory of automaton.

It is appropriate, at this point, to introduce at least part of the syntax of propositional dynamic logic. For convenience, let lower case letters at the beginning of the alphabet denote programs (defined below) and let lower case letters from p on denote (well-formed) formulae of propositional dynamic logic. The basic class of well-formed formulae is denoted Σ_{PDL} and is defined as follows. We start with a countable set of atomic formulae, denoted by Σ_0 , and build inductively: atomic

formulae are formulae; $\perp$ is a formula; if p and q are formulae and a is a program then $\neg p$, $p \rightarrow q$ and $\langle a \rangle p$ are formulae. Formulae that are not atomic are called compound formulae.

The language we use to build programs is based initially on the following: (NOTE: regular PDL and variations)

- i) The language shall include a set of names that denote a set of previously defined primitive or basic programs. This set of primitive program names is called Π_0 . Since we cannot directly support variable assignment, for example, we could elect to make it primitive ($x \leftarrow 0$, $x \leftarrow x+1$, etc.). From Π_0 we can inductively construct the compound programs according to the remaining rules.
- ii) The language shall include special symbol λ . λ denotes the constant identity program that does nothing but produce the same output from its input.
- iii) The language shall support the concatenation of two (or more) programs to create a new program. The operation of concatenation is denoted by the symbol ";". Thus, if *IncrementByTen* and *DecrementByTen* are programs that output values that are 10 more and 10 less than their original input respectively, then *IncrementByTen;DecrementByTen* is a program that, in terms of numerical input/output, acts as λ .
- iv) The language shall support the nondeterministic choice or branching of executing either of two programs. The operation of nondeterministic choice is denoted by the symbol "+". Thus, *IncrementByTen+DecrementByTen* is the program that executes either one of its subprograms *IncrementByTen* or *DecrementByTen* - the choice being made nondeterministically.
- v) The language shall support the iteration of a program a nondeterministic finite number of times. The iteration operator is denoted by the symbol "*". Thus *IncrementByTen** is the program which executes *IncrementByTen* a finite (but

nondeterministic) number of times producing as output some multiple of ten of the numerical input.

The well-formed programs established by these rules is denoted Π_{PDL} . Of these rules, only ii) and iii) are strictly deterministic (i.e., they do not introduce any additional nondeterminism). It is possible to define a dynamic logic that considers only deterministic programs. However, in this thesis we treat nondeterminism as fundamental. We defer a discussion of deterministic dynamic logic and the virtues of nondeterminism until section 1.8.

The program expressions allowed by the five rules given above are very rudimentary and do not capture many popular programming language structures. The real expressive power of a logic is in its ability to capture the detail of the semantic domain. Each of the extensions that we shall consider in this thesis genuinely adds to the expressive power of dynamic logic - they are not just abbreviations of complex expressions made up from the other rules. The stepwise increase of expressive power is the main criterion for our choice of extension. (NOTE: shuffle operator) Our choice of extension, therefore, is based on two reasons: the ability of it to define many well known and popular constructs of the programming language discipline and in its ability to increase the expressive power of dynamic logic. We deal with the first of these criteria here. The latter will be formally verified in chapter five.

The first, and most necessary extension is the ability to test conditions. Our extension will allow us to define the various structured branching and iteration constructs such as the standard *If . . . then . . . else . . . endif*, *While . . . do . . . enddo* and *Repeat . . . until*. As we shall soon see, this simple extension has extremely powerful expressive quality. In order to define a test construct we must incorporate a way to assert conditions within the definition of program. That is, the operator takes a condition (asserted by, say, some propositional calculus) and creates a program element. In this thesis, any well-formed formula of the language of dynamic logic can be a condition for testing. (NOTE: limited tests)

- vi) Let p be a well-formed formula of our language, then $p?$ is the *programmatic test* of the truth of p . The program $p?$ will terminate only if p is true. If the proposition p is false then $p?$ will not terminate (i.e., abort - in practice). (NOTE: side effects of $?$)

Thus, programs will be allowed to test arbitrary formulae of dynamic logic when these formulae involve program expressions. For example, the somewhat redundant formula $\langle \langle a \rangle p \rangle?; a \rangle p$ is permitted. In a nutshell, our programs have the power to "pause" during their execution to make queries of other programs without actually incurring their side effects (i.e., testing a program without running it!). Knowledge-oriented and declarative or goal-oriented programming languages (such as **PLANNER** and **Prolog**) and expert systems require this type of rich exploratory test.

Now it is a straightforward matter to build important *structured* programming language constructs:

$$\begin{aligned} \text{if } p \text{ then } a \text{ else } b \text{ endif} &=_{\text{def}} ((p?);a)+((\neg p?);b), \\ \text{while } p \text{ do } a \text{ enddo} &=_{\text{def}} (((p?);a)^*)(\neg p?), \\ \text{Repeat } a \text{ until } p &=_{\text{def}} a;(((\neg p?);a)^*);p?. \end{aligned}$$

Although it is rare as a programming language construct, the converse operator (first introduced in section 1.2) is useful for reasoning about programs. In backward reasoning one inquires about starting (or pre-) conditions that bring about a given termination (or post-) condition of a program. Obviously, backward reasoning is necessary for diagnostic expert systems where one asks how a set of "symptoms" could have come about. (NOTE: uses of converse)

- vii) The backwards execution of a program a is denoted a^c .

Another useful propositional notion in computer science is that of infinite execution (a subspecies of which would be infinite looping).

- iix) Let the symbol *repeat* denote the predicate which, when applied to a program, expresses the fact that the program can be executed repeatedly without ever terminating. Thus *repeat*(a) for some program a, says that program a* can diverge or enter into a non-halting computation.

The *repeat* operator doesn't capture the idea that a program may be stuck in a non-halting computation (by, say, executing an infinite loop). Another equally well studied construct for dynamic languages is the *loop* operator. Let the formula *loop*(a) indicate that a program a (not a*) can enter a non-halting computation (possibly an infinite loop). *loop*(a) can be defined inductively when the *repeat* operator is present as follows:

$$\begin{aligned} \text{loop}(A) &=_{\text{def}} \perp \text{ for } A \in \Pi_0, \\ \text{loop}(\lambda) &=_{\text{def}} \perp, \\ \text{loop}(a+b) &=_{\text{def}} \text{loop}(a) \vee \text{loop}(b), \\ \text{loop}(a;b) &=_{\text{def}} \text{loop}(a) \vee \langle a \rangle \text{loop}(b), \\ \text{loop}(a^*) &=_{\text{def}} \text{repeat}(a) \vee \langle a^* \rangle \text{loop}(a), \end{aligned}$$

and when ? or ^c are part of the language then,

$$\begin{aligned} \text{loop}(p?) &=_{\text{def}} \perp, \\ \text{loop}(A^c) &=_{\text{def}} \perp \text{ where } A \in \Pi_0. \end{aligned}$$

Therefore, the *loop* operator does not add to the expressive capabilities of a dynamic logic that supports *repeat*. When *repeat* is not available, however, *loop* does genuinely enhance the expressive power of propositional logic. (NOTE: definition and expressive power of *loop*)

Perhaps a more applicable divergence predicate would be one directly capturing the notion of infinite loop, that is, the ability of a program to execute forever by repeatedly executing some part of itself over and over again, ad infinitum. Let *repeat-cycle*(a) express that the program a* can enter into an infinite loop. Also, let *loop-*

cycle be the predicate, analogous to the *loop* operator, which says that its program argument can enter into an infinite loop. We can define *loop-cycle* inductively by using *repeat-cycle* in the same way that *loop* was defined in terms of *repeat*. Clearly, if *repeat*(a) then *repeat-cycle*(a) - semantically *repeat*(a) whenever if a^* is started then it may continue indefinitely and *repeat-cycle*(a) whenever a^* is started then there is some finite sequence of executions of a can be repeated indefinitely. (NOTE: *cycle*)

This thesis limits its consideration to the matrix of programming languages defined by i) to v), supplemented by some of vi) to iix). Chapter two formally defines these languages and builds the well-formed expressions (both programs and formulas) of the various logics based on these languages. Axioms and rules of inference are then introduced to enable us to reason about program expressions. Chapter three focuses on the denotational "meaning" of well-formed programs and on the validity of expressions about them. To this end, several different classes of models are distinguished - most of which are considered non-standard. The existence of appropriate and manageable models for the various propositional dynamic logics is discussed. Also, in chapter three, such semantical issues as compactness and the complexity of deciding whether a formula is valid or not, are investigated. The non-trivial issue of completeness for our logic is reserved for chapter four. Finally, chapter five compares the various logics on the basis of their respective expressive capability.

Not considered, as mentioned at the onset of this section, are many programming language constructs that are deservedly thought of as fundamental. Perhaps the most important of our omissions is the capability for general variable or array assignments. The implementation of general variable assignment is beyond the scope of propositional logics requiring additional objects - terms and variables of different types, arrays, etc. - that can only be handled by quantificational dynamic logic. (NOTE: 1st-order dynamic logic) Nor does our programming language support recursion (NOTE: recursion) or calls to subroutines. (NOTE: subroutines) Many, less complicated but also less "mainstream" propositional level extensions have also been omitted. (NOTE: other extensions)

1.7 Applications of Dynamic Logic

Other than using dynamic logic as a logic for reasoning about (and mechanically verifying) computer programs (NOTE: Goldblatt's work) and actions in general (NOTE: Segerberg's work), it has been indicated in the area of natural language processing. It is difficult - probably impossible - to see a natural language as a formal language within the context of the classical "static" logic. It is less so in the context of dynamic logic. For example, the ; operator can be seen as a type of conjunction where order counts i.e., ; is not commutative. For example, this is helpful when breaking down the proposition

After you open the door and walk through it you are in the room
into obvious constituent parts. In the formalism of dynamic logic, the above proposition is equivalent to

<open the door;walk through the door>you are in the room.

Another simple example illustrates the convenience of iteration. In dynamic logic it is possible to show that the proposition

If your T.V. won't work then no matter how many times you kick it, it still won't work
follows from the basic step

If your TV won't work and you kick it, it still won't work.

(NOTE: Pratt's example #2)

Perhaps the ultimate application of dynamic logic will be to the brain - the materialist/reductionist paradigm of intellectual machinery. A prerequisite to application must be a deeper understanding of the brain. Currently, the most widely regarded models of the brain equate it with a highly complicated computer, whose moving parts or "switching elements" are the approximately 10 to 100 trillion neural synapses. If the state of an electronic computing machine is comprised of the on/off configuration of its switching elements, then the corresponding definition of a brain state is the total condition of the brain's synapse. According to this definition there are 2^n possible brain states where n is the total number of synapses. This characterization would explain the deep complexity of the human brain. Mapping accessibility between these states for any given "thought" would be an enormous task.

Perhaps the only barrier to applying dynamic logic to brain is this complexity.(NOTE: Sagan)

1.8 Determinism Versus Nondeterminism

Almost all programming languages allow only deterministic programs and the bulk of all computer programs give wholly deterministic solutions to problems.(NOTE: logic programming) This is good support for considering the more restrictive deterministic version of programs. Outside the realm of computer science, on the other hand, and complementing its greater theoretical interest, nondeterminism offers the possibility of a wider range of application. Indeed, if dynamic logic is ever applied as a logic concerning the accessibility relationships between brain states, then the impasse of the determinism/free-will debate supports the more encompassing view. Moreover, a practical logic of general action seems to favour (and even demand) this type of nondeterminism.(NOTE: soft determinism) (Far beyond this, support for a nondeterministic view of the "real world" is supported at the subatomic level by quantum theory!)

But even within the scope of computer science there are strong reasons for studying nondeterministic programs.(NOTE: support for nondeterminism) The most obvious reason is that nondeterminism can be used to model concurrency (such as that found in timesharing operating systems where "nondeterminism expresses the apparent capriciousness of the scheduler") and distributed processing (where, for example, "nondeterminism captures the unpredictable behaviour of physical devices"). Another area where nondeterminism is proving useful is in AI. Take, for example, the technique of "generate-and-test" employed by the AI language **PLANNER** and other fourth generation computer languages. A **PLANNER** program consists of *GOAL-statements* with no explicit solutions. A **PLANNER** programmer is not unduly concerned about the way a goal is reached - **PLANNER** resolves the problem itself by generating potential solutions and then testing them for validity.

Even if all programs were deterministic it is still often more appropriate to reason about them nondeterministically. To take a simple example, one way to argue that the execution of the deterministic program

if $x > 0$ then $x \leftarrow x - 1$ else $x \leftarrow x + 1$ endif

does not affect the variable y is to show that execution of either of its consequent clauses will not affect y - that is the execution of the nondeterministic program

$(x \leftarrow x - 1) + (x \leftarrow x + 1)$

cannot change y . Similarly, demonstration that the deterministic program

while $x < 0$ do $x \leftarrow x + 2$ enddo

does not alter the parity of x depends essentially on the observation that the execution of nondeterministic program

$(x \leftarrow x + 2)^*$

will not change the parity of x .

Because of its predominance in computer programming we now indicate how nondeterminism can be eliminated from regular dynamic logic. There are two approaches to this problem. One approach is wholly semantical, confining the semantics to deterministic models. The other approach, which is mainly syntactical, disallows any program construct that may introduce nondeterminism. We deal with the semantical approach first.

In dynamic logic nondeterminism is introduced in two ways. First of all there may be non-deterministic basic programs (interpreted as relations that are not necessarily partial functions). This type of nondeterminism can be easily excluded by ensuring that basic programs are interpreted by partial functions. Nondeterminism may also be built into compound programs with the operators $*$ and $+$. By modifying the interpretation of these operators so that they take only partial functions to other partial functions of the state domain we ensure that compound programs are given a strictly deterministic interpretation. (Notice that the version of $Ax.det$ for dynamic logic, i.e., $\langle a \rangle p \rightarrow [a]p$ is now a valid formula schema with respect to the above semantics). Propositional dynamic logic confined to partial functions is called deterministic propositional dynamic logic. (NOTE: DPDL)

Another technique for excluding nondeterminism from compound formulae is to eliminate those well-formed programs that *can* introduce nondeterminism. Therefore the programs $a;b$, a^* and others are removed and replaced by deterministic operators. In fact, the well-formed programs of strict or structured propositional dynamic logic only have the nondeterministic operators of $;$ and $*$ appearing in such purely deterministic and structured programming constructs as *if-then-else-endif*, *while-do-endo* and *repeat-until* defined in section 1.6. Unfortunately, this method has no way of preventing basic programs from being interpreted by nondeterministic relations. Thus, in order to arrive at strict deterministic propositional dynamic logic we impose the semantic restriction that all basic programs be interpreted by partial functions. (NOTE: SDPDL)

Chapter 1 Notes:

Thompson

The term *exercise of agency* originates (I think!) from Judith Jarvis Thompson's book *Acts and other Events*, Ithica (1977). Although Thompson's use of the term is a little more restrictive than our own, a quote from this book will help to clarify: "What I have in mind is the thought that a man's killing of a man is an 'exercise of agency', whereas an inanimate object's killing of a man is not. What is an 'exercise of agency'? A 'thing done', an event with an 'agent'. A man (it is said) may be 'agent of' a killing; an inanimate object - an event, bullet or wound - may perhaps kill a man, but is not an agent of any killing." [*ibid*, pp.24] We would rather say that an action involves an animate agent, but that other *agents of exercise*, such as computer programs, may be inanimate.

name

The name "dynamic logic" was suggested in [HMP77] and is credited to Vaughan Pratt, the inventor of dynamic logic, by Segerberg in [SEG80]. Prior to this paper the most popular name for dynamic logic was the "modal logic of programs" (especially in the early papers on propositional dynamic logic). The connection with modal logic will become obvious as we proceed.

theoretical interest

It turns out that several, if not all, versions of first-order predicate dynamic logic and many other first-order systems for reasoning about programs can be embedded in the infinitary logic $L_{\omega_1\omega}$, and so, presumably, it is of less theoretical interest (in its own right) than it would have otherwise. "These systems can be described in more classical terminology as fragments of the constructive portion of $L_{\omega_1\omega}$, with different instances of the systems characterized by various simple syntactic conditions on infinitary formulas [*sic*]." [M&T81;pp.285] Thus, Albert Meyer has been inclined to say that "... there doesn't seem to be anything dynamic about Dynamic Logic anymore ... we find ourselves doing what logicians did twenty years ago, when they first got interested in infinitary logic." [X81;pp.409] "The particular set of questions involving the first order extensions of Dynamic ... Logic ... have reduced to some clear logical questions about $L_{\omega_1\omega}$." [*ibid*;pp.415] First-order dynamic logic is still of great interest to the computer scientist, and it is, in many cases, just because of this correspondence [*ibid*; see remarks of Harel, Parikh and Nemeti pp.409-416]. I find Meyer's criticism untenable: Many logics are "theoretically" interesting just because they are subsets of more powerful systems (e.g., intuitionist logics). He also indicates that propositional dynamic logics are more interesting than their quantified counterparts and this would seem to contradict his earlier position: "Since the propositional dynamic logic can be embedded in its first-order version, isn't it of less theoretical interest?"

theoretical inquiries

Apart from its place as a species of mathematical logic, and its more obvious

applications, dynamic logics have been found to be closely related to admissible sets (Ershov, Y.L., *Dynamic Logic over Admissible Sets*, Dokl. Akad. Nauk USSR, Vol. 273 (1983), pp.1045-1048). In its algebraic form, dynamic logic has been used to study the nature of induction [PRA80] and in defining new axioms for regular algebras (a la Conway, J.H., *Regular Algebra and Finite Machines*, London (1971)). (In the latter context, we note that propositional dynamic logic is finitely axiomatizable despite the fact that an embedded fragment of it, the equational theory of regular algebra, is not (see V.N. Redco, *On Defining Relations of Regular Events* (in Russian), Ukrain. Mat. Z., Vol.16, pp.120-126 (1964)).

verifier

"From a practical point of view [dynamic logic offers a formalism that] already exceeds the level of detail a modern program verifier would demand to be persuaded of the soundness of an argument. A sensible approach in building such a verifier is to have a general purpose algorithm for testing whether each inference in a proof is sound, rather than whether the proof fits the axioms and rules of some axiomatization of the logic. Such an approach permits the user of the verifier to supply shorter proofs - just how short depends on how good the soundness checker is." [PRA80b;pp.269] This will not be a topic that we take up in this thesis. We just note that at least one verifier, utilizing dynamic logic existed in 1977. In particular, see S.D. Litvinchouk, and V.R. Pratt, *A Proof-checker for Dynamic Logic*, *Proceedings of the 5th International Joint Conference on Artificial Intelligence, 1977*, which reports on a verifier running at M.I.T. "... incorporating early ideas about D[ynamic]L[ogic], and implemented in LISP on a PDP-10." [PRA78b;pp.21] "... the largest ... proof that has been subjected to successful mechanical verification to date is a proof of the total correctness of the Knuth-Morris-Pratt pattern-matching algorithm ... The proof consisted of some twenty theorems stated in D[ynamic]L[ogic], and was certified in 45 seconds." [ibid.]

decidability

[PRA78b] reports the discovery of improved methods for deciding the satisfiability and validity for formulae of dynamic logic - which would (and perhaps did) lead to significant improvements in the verifier of the previous note. Indeed, the motivation for studying decidability in this case was to improve the verifier.

Pratt's example #1

This example is from [PRA81b].

total correctness assertions

Notice that $p(a)q$ does not assert the termination of a , hence the designation *partial*. If precondition p happens to be false, a may never terminate. Total correctness, on the other hand, refers to an assertion that a program terminates and is (partially) correct.

Floyd-Hoare history

The Floyd-Hoare logic began in 1967 with R.W. Floyd's attempt at proving correctness of *flowcharts* (*Assigning Meanings to Programs*, Proc. of the Symp. of Applied Math., J.T. Schwartz (ed), Vol. 19, pp.19-32). Assume that all flowcharts under our consideration have unique entry and exit arrows and attach well-formed formulae of $\mathcal{L}_{CQL}$ to these entry and exit arrows (these are the precondition and postcondition respectively). For reasons that will become clear in section 1.3 this pair of CQL assertions is called an "interpretation". An "interpretation" is said to be correct if the postcondition is true when the exit point is reached, provided that the precondition was true on entry to the flowchart. Floyd provided four rules for establishing correctness. Correctness of a flowchart algorithm is then reduced to showing that the intended "interpretation" satisfies these rules. In 1969 C.A.R. Hoare introduced the compact notation for assertions about flowcharts that we see here (*An Axiomatic Basis for Computer Programming*, Communications of the ACM, Vol.12, pp.576-580). Floyd's four correctness rules for PCA's can be written with Hoare's notation as follows:

Rule of assignment: $f(x/y)\{x \leftarrow y\}g$

where $f(x/y)$ is the assertion obtained by replacing all free occurrences of x in g by y ;

Rule of consequence: $p(a)q, p' \rightarrow p, q' \rightarrow q$ implies $p'(a)q'$;

Rule of iteration: $(p \& q)\{a\}r$ implies $p(\text{while } q \text{ do } a \text{ enddo})(r \& \neg q)$;

Rule of composition: $p(a)q, q(b)r$ implies $p(a;b)r$;

where $a;b$ denotes the program that executes a followed by b .

Subsequently Hoare (and collaborators) added the following rule:

Conditional rule: $(p \& q)\{a\}r, (p \& \neg q)\{b\}r$ implies $p(\text{if } q \text{ then } a \text{ else } b \text{ endif})r$

operational semantics

"This [view] is a natural development of the original view of high-level languages, which was as a kind of short-hand for writing machine code; it was at the machine code level that the real computing was analyzed. (Nowadays we tend to think of high-level languages more on their own terms and at their own level, but the name FORTRAN survives as a reminder of the older view.)" [STR77;pp.19]

success of the axiomatic approach

C. Hoare and N. Wirth (*An Axiomatic Definition of the Programming Language PASCAL*, Acta Informatica, Vol.2, pp.335-355 (1973)) based their partial axiomatisation of PASCAL on the Floyd-Hoare system and extended it to cover operations for various non-trivial data types and procedure/function declarations. This represents the first practical application of Floyd-Hoare logic to an already established programming language. ("The methodological perspective that has emerged from these axiomatic investigations is that the programmer should be encouraged to develop a correctness proof concurrently with the original development of his algorithm, rather than try to reconstruct one afterwards. A textbook has been written [The Design of Well-Structured and Correct Programs by Saud Alagic and Micheal A. Arbib (Springer-Verlag, 1978)] that aims to teach the reader how to use PASCAL in this way." [GOL82;pp.16-17])

S. Cook (*Axiomatic and Interpretive Semantics for an ALGOL fragment*, Tech. Report 79, Dept. of Computer Science, University of Toronto (1975), and *Soundness and Completeness of an Axiom System for Program Verification*, SIAM Journal of Computing, Vol.7, pp.70-90 (1978)) provides a set of finitary rules based on the Floyd-Hoare rules for a deterministic fragment of the programming language ALGOL and shows these to be complete (i.e., all valid PCA's with valid preconditions and valid postconditions are derivable from his rules).

D. Leivant (*Hoare's Logic Captures Program Semantics (Extended Summary)*, unpublished report, Dept. of Computer Science, Carnegie-Mellon University) has shown that the Floyd-Hoare logic does give a complete (in the sense of Cook) axiomatic semantics for While-programs. Whereas Leivant's demonstration does not easily extend to other classes of programs, a simpler proof of this result due to A. Meyer (*Floyd-Hoare Logic Defines Semantics: Preliminary Version*, IEEE Symposium on Logic in Computer Science, Cambridge, Mass (1986), pp.40-48) shows greater promise - "... we expect in the final version of this paper to establish an [adequate] axiomatic semantics for full algol (sic) -like program schemes without restrictions." [ibid;pp.48]

adequacy of Floyd-Hoare logic

Despite the positive results of the previous note, Floyd-Hoare logic is certainly inadequate for all but very simple programming languages. Mitchell Wand (*A New Incompleteness Result for Hoare's System*, JACM, Vol.25 (1978), pp.168-175) gives a PCA which is obviously valid with respect to a specific data type but which is not derivable from the rules of Floyd-Hoare logic (see (NOTE: Floyd-Hoare history)). Whereas Wand's result is with respect to a specific data type, E.M. Clarke (*Programming Language Constructs for which it is Impossible to Obtain Good Hoare Axiom Systems*, JACM, Vol.26 (1979), pp.129-147) gives a more general set of program structures whose presence guarantees inadequacy. Following Clarke's observations, R. Lipton and L. Snyder (*Completeness and Incompleteness of Hoare-like Axiom Systems*, Unpublished manuscript (1977), Dept. of Computer Science, Yale University) and R. Lipton [LIP77] give a simple necessary and sufficient condition for a programming language to ensure the adequacy of Floyd-Hoare logic: it must have a decidable halting problem for finite models.

These results can be circumvented by the addition of the following rule. Let a^n denote the program which is executed repeatedly n times.

The omega-iteration rule: If, $\forall n < \omega, p(a^n) \rightarrow r$ then $p(\text{while } q \text{ do } a \text{ enddo})r$. (Notice that this rule employs infinitely many premises. The situation here is the same as circumventing the incompleteness of Peano arithmetic by the addition of the well-known "Omega rule". See [GOL82;pp.22-23] for more on this.)

Scott-Strachey denotational semantics

The most sophisticated and well-known declarative/denotational semantics is the work of Dana Scott and Christopher Strachey at Oxford University in the late 60's and early 70's. A good introduction to their theory is [STR77].

Leibniz

"It follows from the supreme perfection of God, that in creating the universe he has chosen the best possible plan, in which there is the greatest variety together with the greatest order; the best arranged ground, place, time; the most results produced in the most simple ways; the most of power, knowledge, happiness and goodness in the creatures that the universe could permit. For since all the possibles in the understanding of God laid claim to existence in proportion to their perfections, the actual world, as the resultant of all these claims, must be the most perfect possible. And without this it would not be possible to give a reason why things have turned out so rather than otherwise." From *The Principles of Nature and of Grace*, chapter 32 (pp.209-217) of *The Philosophical Works of Leibniz*, translated by G.M. Duncan, New Haven (1890) (from the original, *Principles de la Nature et de la Grace, Fondes en Raison*).

ontology of possible worlds

Unfortunately, this view begs more questions than it answers - with most of the blame going to possible worlds themselves. "The trouble with possible worlds . . . is that they represent an exotic piece of metaphysical machinery, the armchair invention of a speculative ontologist lacking what Bertrand Russell called 'a robust sense of reality'" (Michael J. Loux, *Modality and Metaphysics*, the introduction to *The Possible and the Actual: Readings in the Metaphysics of Modality*, ed. Michael J. Loux, Ithica (1979) pp.30). See this reference for more details on the various criticisms and defences of possible worlds.

transworld identity

Another serious problem with possible worlds is that of locating individuals in possible worlds (this is called the problem of *transworld identity*). Kripke's response to this problem is of some indirect interest to us. In his famous article, *Naming and Necessity*, Kripke states that "'possible worlds' are stipulated, not discovered by powerful telescopes" (in *Semantics of Natural Language*, ed. D. Davidson and G. Harman, Dordrecht (1972) pp. 267). This comment is certainly not a sufficient defence of possible worlds, but it does suggest less problematic applications of pure modal logic - such as for reasoning about programs (as will become clear as we proceed) and as a logic of *provability* (see [BOO79]). To quote from [SEG80]: "Even a hostile critic of applied modal logic will admit that dynamic logic offers an interpretation of modal logic that is beyond formal reproach - fruitful or not, it is precise and perfectly intelligible." pp.292.

process logic

If, on the other hand, we are also interested in the execution process (i.e., each possible execution path) of a program *a*, then we should continue to interpret *a* as a set of sequences of states that can be generated by executing *a*. Process logic (invented, once again, by Pratt ([PRA78] and [PRA79]) develops propositional dynamic logic to the point of being able to reason about a program during its execution. The following modalities are added to dynamic logic (the notation here is from [SEG80]):

- <<a>> - "sometime during at least one execution of a . . .",
- <<a]] - "throughout at least one execution of a . . .",

The duals of these modalities are, respectively:

$[[a]]$ - "throughout every execution of a . . .",

$[[a]>$ - "sometime during every execution of a . . .".

(For obvious reasons Pratt refers to these latter modalities as "a maintains . . ." and "a promises . . .".) Another useful, albeit complicated modality, formalizes the notion of preservation:

$\text{pres}_a p$ - "if p is ever true during the execution of a then p will remain true".

[PRA78], [PRA79], [SEG80] and other papers by Pratt and Segerberg give partial completeness for a finite axiomatisation of the above logic. Parikh [PAR78a] gives a decidable process logic, called SOAPL, extending that of [PRA79] so that it includes quantifiers that range over states and execution paths. SOAPL is strictly more expressive than [PRA79] since it is capable of expressing Pnueli's temporal connective $\Psi(a,p,q)$ (meaning that "during any execution of a , if p is ever true then q is also true at some time later"). The powerful process logic, NL, proposed by Nishimura [NIS80] combines Pnueli's temporal logic for reasoning about concurrency of a single assumed program ([PNU77], [PNU79] and also [M&P79]) with propositional dynamic logic. NL subsumes SOAPL and, hence, also Pratt's version of process logic. The best candidate for an "official" process logic, however, is the system PL proposed by Harel, Kozen and Parikh [HKP80]. PL, like NL, combines the temporal logic of Pnueli with propositional dynamic logic. PL extends propositional dynamic logic to include the following temporal modalities:

f - "at the beginning . . .",

$\text{su}f$ - "until . . .".

(The former of these is a unary connective i.e., $f p$ means "at the beginning p is true"; the latter is a binary connective i.e., $\text{psu}f q$ means " p is true until q is true".) PL completely subsumes all of the aforementioned process logics and keeps as axioms the traditional axioms of propositional dynamic logic (the so-called Segerberg axioms, cf. chapter two, section 2.3). [HKP80] gives a finite axiomatisation of PL which is both complete and decidable. These positive results, however, should be tempered with the fact that PL is no more expressive than propositional dynamic logic when basic programs are interpreted by binary relations and when all execution paths are of finite length. (This observation is reported in [HAR84;pp.597].)

Notice that process logic also answers the problem of extensionality in dynamic logic i.e., the identification of two different programs with the same denotation. Consider, for example, the identity program and a program that, say, increments a variable by ten and then immediately decrements the same variable by ten. Although they are very different programs, in dynamic logic they are interpreted by the same binary relation. In process logic, however, the above programs are distinct since all component states of all execution sequences must correspond in order for programs to be identical.

modal logic

The beginnings of modal logic can be traced as far back as Aristotle and Philo of Megara in the fourth century B.C., and was popular among Arab and Christian

scholastics in medieval periods. However, it was not until this century that C.I. Lewis cast modal logic in its modern symbolic form (see especially C.I. Lewis and C.H. Langford's *Symbolic Logic*, New York, 1932, 2nd edition, 1951). (Incidentally, Lewis resurrected modal logic in an attempt to resolve the paradoxes of material implication.) After Lewis, it is generally accepted that the single most important development in modal logic was Kripke's definition of *necessary truth* in terms of possible worlds, and therefore his completion results for Lewis' formal systems [KRI63].

Historically modal logic is an attempt to formalize the concepts of *necessity* ($\Box$), and *possibility* ($\Diamond$). Also, attempts have been made to formalize such diverse concepts as *time* (vis a vis tense logics) and *moral obligation* (vis a vis deontic logics) in the same fashion. These formalizations are now considered to be within the subject matter of pure modal logic. Neither is it required that the number of additional connectives be limited to one. For example, attempts have been made to mix different species of modalities (e.g., contingency versus logical possibility) in one consistent system. Thus, we regard pure *modal logic* as the logic of non-truth functional connectives. (Which implies, incidentally, that dynamic logic is a form of modal logic.) For more information on pure modal logic consult [CHE80], [LEM77] or [SEG71]. [LEM77] and [SEG71] are considered classics in this field.

correspondence theory

The study of the connection between intensional axioms, such as those of modal and dynamic logic, and the properties of the accessibility relation has been dubbed correspondence theory. This has been an intensely active area of research since Kripke introduced his relational frame semantics in [KRI63]. See [VB84]. In light of the above queries, modal correspondence theory would seem to be of special importance to computer scientists. Indeed, the incompleteness of certain modal logics with respect to Kripke frames may eventually answer the question of the adequacy of the axiomatic approach to program semantics.

regular PDL and variations

Notice that, in the following definition, if Π_0 were finite, then rules i) through v) produce programs that are *Kleene regular expressions* over the alphabet Π_0 . This precise notation is the invention of Stephen Kleene (*Representation of Events in Nerve Nets and Finite Automata*, *Automata Studies*, Princeton University Press, Princeton, New Jersey, pp.3-42 (1956)). It is well known that any regular expression "denotes" a set, referred to as a regular set or language, that is accepted by a finite automaton. Thus, programs based on i) through v) denote regular sets (sometimes called regular languages) of finite execution sequences of atomic programs. For this reason, the dynamic logic based on programs produced from these rules is called the dynamic logic of regular programs (even though Π_0 is in general countable rather than finite). This suggests an extension of programs to other languages of the Chomsky hierarchy. One variant of dynamic logic takes programs to be context-free sets of execution sequences. The obvious motivation for context-free dynamic logic is the well-known correspondence between recursive programs (i.e., pushdown automata) and context-free

languages. "Moreover, one is not limited merely to context-free programs. The most generous class of programs to consider is that of *r.e.* programs; i.e., recursively enumerable sets of execution sequences. These are programs which might require the full power of effective computations (via unrestricted Turing machines) just in order to determine the next-step relation which controls the flow of execution. In contrast, regular and context-free programs require, respectively, the power of finite automata and pushdown automata." [HAR84;pp.528]

Unfortunately the dynamic logic of context-free programs is highly undecidable - a result that follows from the unsolvability of the equivalence of representations for context-free languages. This should be contrasted with the single exponential deterministic time solution of the validity problem for the dynamic logic of regular programs (see chapter three, section 3.5). Indeed, the inclusion of the single context-free program $A^{\Delta}BA^{\Delta}$ ($= \bigcup_{i \geq 0} A^iBA^i$ where $A^i = A;A; \dots ;A$ i.e., the concatenation of i A's as defined below in step iii) for some fixed basic programs A and B in the set of regular programs, is enough to make propositional dynamic logic highly undecidable [HPS81a]. Further undecidability results for context-free programs can be found in [HPS81] and [HPS81a]. There are, on the other hand, context-free programs (indeed, whole families of them) whose addition to the set of regular programs preserves the decidability of the validity problem of propositional dynamic logic [KP83].

Another approach connects dynamic logic directly to varieties of automata rather than through their corresponding "intermediary" Chomsky formal languages. This variation has gained considerable attention since first advocated by Pratt in [PRA81c]. In part, this is because automata can provide an economical and "natural" representation of programs. His propositional *flowGraph logic*, or PFL, takes programs to be finite automata instead of regular expressions. "Our primary interest in PFL is its apparent greater succinctness; of secondary interest is the insight that PFL gives [propositional dynamic logic]" [PRA81c;pp.389]. Many new results about dynamic logic are being inferred through this approach (e.g., the exponential decidability results and completeness of propositional dynamic logic with *loop* due to Pnueli and Sherman (*Propositional Dynamic Logic of Looping Flowcharts*, Technical Report, Weizman Institute, Rehovot, Israel, 1983), and the decidability results of Stockmeyer, Vardi and Wolper concerning propositional dynamic logic with *loop* or *repeat* supplemented with converse (see [VAR85] for details). For details on this approach to dynamic logic consult [PRA81c] and, especially, [H&S85] where PFL is referred to as APDL.

shuffle operator

Therefore, because it is definable in nondeterministic propositional dynamic logic with tests (reported in [HAR84;pp.535]), Abrahamson's shuffle operator [AB79] is not considered. The shuffle operator, denoted $//$, is useful for reasoning about concurrent programs: the compound program $a//b$ denotes the interleaving of execution sequences of a with those of b .

limited tests

The dynamic language without programmatic tests, however, does provide for something in the way of testing (called *ground testing*). This is clear from the test axiom (which is formally introduced in chapter two) $Ax.?: \langle p? \rangle q \equiv p \& q$. This implies that the propositional operator $\vee$ (which, with $\neg$, defines the $\&$ -operator) may be eliminated without the loss of expressive capability when tests are included in the logic. The $?$ operator, on the other hand, cannot be eliminated with a cost in expressive power. This will be demonstrated in chapter five.

side effects of ?

We assume that the actual programmatic testing of p has no side effects - it will not cause a change in the "state" of the machine. It can only block or allow the continuation of a program of which it is part.

uses of converse

In addition to expert systems, backward reasoning is employed by declarative programming languages (e.g., Prolog, PLANNER, etc) and constraint-propagating networks.

definition and expressive power of loop

In chapter two (theorems 2.5.13 to 2.5.17 and 2.5.22 to 2.5.26) we will find that the $\circ$ operator distributes over all of the other operators. Thus, when defining $loop(a)$ we need only consider the case where a is a basic program. This construct first appears in [H&P78] and is denoted by $\langle a \rangle^+ \perp$. Later Pratt adopted the notation ∞a and $loop_a$ for $loop(a)$ [PRA80b; pp.268-269] (which is also used in [STR82; pp.125]). Our notation is from [H&S82]. V. Pratt [PRA80b; pp.269] observes that $loop$ genuinely adds expressive power to propositional dynamic logics that lack *repeat*. D. Harel and R. Sherman prove that propositional dynamic logic with tests and *repeat* is strictly more expressive than propositional dynamic logic with tests and *loop*.

The expressive power of 1st-order dynamic logic is not enhanced by the *loop* operator. This surprising result is demonstrated by a complicated argument in [M&W82].

cycle

These predicates are related to the *cycle*-predicate of R. Danecki (*Propositional Dynamic Logic with Strong Looping Predicate*, Proc. of Mathematical Foundations of Computer Science (1984), LNCS 176, pp.573-581) and M.Y. Vardi and P. Wolper (*Automata Theoretic Techniques for Modal Logics of Programs*, Proc. ACM Symposium on Theory of Computing, Washington, April 1984, pp.446-456). See also [VAR85]. In these, $cycle(a)$ is true when program a can return to its starting state. Propositional dynamic logic, as given by rules (i) through (iix) but with *loop* instead of *repeat*, and extended to support *cycle*, is undesirable. (This is shown in the Danecki paper.)

1st-order dynamic logic

First-order dynamic logic or QDL (for quantified dynamic logic), which deals with substitutional terms of a programming language, was first introduced in the

foundational work of Pratt [PRA76] (i.e., QDL preceded its propositional counterpart as introduced by [F&L77]). QDE may be seen as an enhancement of propositional dynamic logic that allows variable assignment. Assuming a countable set of variable symbols and some standard definition of *terms* (e.g., [HAR84;pp.546]) then, for variable x and term t , $x \leftarrow a$ is called a *specific assignment*. Another form of assignment due to Pratt [PRA76], is called *random assignment* and denoted $x \leftarrow \text{random}$ (or $x \leftarrow ?$). The semantics of both these types of assignment insist that the only side effect of their execution is a change to the value of the variable being assigned, and no others. For specific assignment, the semantics must also ensure that the value of the assigned variable is the same as the term assigned. For random assignment *any* term value may be assigned to the target variable. When random assignment is added to QDL (with an appropriate axiomatisation) then the resulting logic admits the elimination of the standard first-order quantifiers:

$$\begin{aligned}\forall x \varphi &=_{\text{def}} [x \leftarrow \text{random}] \varphi, \\ \exists x \varphi &=_{\text{def}} \langle x \leftarrow \text{random} \rangle \varphi,\end{aligned}$$

(therefore these quantifiers do not add to the expressive power of QDL with random assignments. On the other hand, QDL with random assignments is strictly more expressive than QDL without them (reported in [HAR79;pp.19]).) QDL may also be extended to support the assignment of a term to an element of an array. The key factor here is to treat an array, say f , as a function mapping an index to a set of values of the appropriate type. Now let $f \leftarrow f_i^t$ where f_i^t denote the new configuration (i.e., function) of the array f that results from assigning the term t to f at index i . As in the case of random assignment, QDL with array assignment does add expressive power (reported in [HAR79;pp.19-20]).

The validity problem for QDL has been investigated in [HMP77] and has been found to be highly undecidable - to the point of preventing any finite axiomatisation. In [HAR78] and [HAR79], D. Harel gives a simple axiomatisation of QDL which is then shown to be arithmetically complete. That is, within a suitable model which contains the natural numbers and the capability for encoding finite sequences, any QDL formula is equivalent to some CQL formula. (Another approach would be to add infinitary rules to QDL similar to those of the infinitary logic $L_{\omega_1\omega}$ e.g., if $p, [a]p, [a][a]p, \dots$ etc, then $[a^*]p$ [PRA80b;pp.266]. [GOL82] takes a similar approach, championing an "omega-iteration" rule (see NOTE: adequacy of Floyd-Hoare logic).)

recursion

Recursion (and iteration (step v, below)) may be supported in dynamic logic by the addition of the *least fixed point* or μ -operator. This construct is inspired by Kleene's first recursion theorem (see [CUT80;chap.10]). Briefly, a recursive program, say a , is one whose expression in a programming language includes reference to itself i.e., $a =_{\text{def}} \tau(a)$ where τ is a definition of the program a , and $\tau(a)$ means that the name of a (the symbol a) occurs in the expression τ . The notation $\mu a. \tau(a)$ is used to denote that $a =_{\text{def}} \tau(a)$. τ gives rise to a "continuous/monotonic" operator on binary relations, say r' , whose least-fixed point is the proper interpretation of the program a . In [HAR79;chap.4] a dynamic logic of elementary recursive programs (e.g., no parameter passing) based on the μ -

operator is considered and completeness in an arithmetical universe is demonstrated.

This method is the same as in the μ -calculus advocated by D. Scott and J. de Bakker (*A Theory of Programs*, unpublished notes, IBM, Vienna (1969)) and developed by many (including P. Hitchcock and D. Park (*Induction Rules and Termination Proofs*, CALP (1973)) and J. de Bakker (*Mathematical Theory of Program Correctness*, Prentice-Hall, Englewood Cliffs, N.J. (1980)). Exponentially decidable propositional level μ -calculus (closely akin to and in the notation of dynamic logic) have been developed by Pratt [PRA81] and Kozen [KOZ82, K&P83]. Kozen's version is also shown to be complete. Both of these logics totally subsume propositional dynamic logic with tests and converse, but only Kozen's version is capable of expressing *repeat(a)*.

An alternative (and equivalent) approach to recursive programs in dynamic logic is to consider context-free programs (see NOTE: regular PDL and variations, above).

subroutines

A *subroutine* (aka *subprogram*) is a part of a program which (through its declaration) has been assigned a name, allocated local variables and whose access to variables outside its definition (such as global variables, arguments passed by reference ("call by name") and/or return parameters) is often strictly controlled. Subroutines may be conveniently invoked from several places (thereby eliminating the need to duplicate parts of a program). Because of its usefulness, such a facility is supported by the most primitive programming languages (i.e., assemblers). For a QDL-like implementation of subroutines (of both flavours, functions and procedures) and the syntax of their invocation see [GOL82; chap.'s 4 & 5, pp.207-250]. See, however, NOTE: Goldblatt's work.

other extensions

The best known of these neglected additions are the intersection and complement operators (interpreted as the set theoretic intersection and complement of the relations interpreting program arguments). See [HAR84; 535-539], [HAR83] and [DAN84] for details.

In [T&V84], T. Tinchev and D. Vakarelov consider two extensions to propositional dynamic logic with tests. First they consider adding special primitive programs to increment and decrement reserved numeric variables, called counters. The other extension requires a battery of special programs that assign specific values to a single reserved variable, called a stack. Both resulting logics are given complete axiomatisations and are shown to be highly undecidable.

S. Radev [RAD84] considers infinitary combinatory propositional dynamic logic, a propositional dynamic logic of regular programs with infinite conjunctions and alterations, and propositional constants. This logic is given an axiomatisation which is shown to be complete with respect to countable Kripke models.

Interesting propositional dynamic logics that are able to assign the value $\top$ or $\perp$ to propositional variables are investigated by M. L. Tiomkin and J. A. Makowsky in [T&M85] (developing and refining some ideas of A. Meyer and K. Winklmann (*On the Expressive Power of Dynamic Logic*, STOC 1979, pp. 167-175)). They consider two types: global assignments simultaneously set a propositional variable for all states; local assignments set the propositional variable only in the current state. Local assignment is a powerfully expressive notion providing near first-order expressive power, but global assignment proves to be of little value.

syntax history

With the exception of *repeat* each of the main constructs - and nearly all of the rest (unless noted) appear in [PRA76]. Tests, converse and iteration appear there for first time. These facts testify to the foundational importance of that paper.

Goldblatt's work

The most ambitious work (in terms of supporting a realistic programming language) of our bibliography is [GOL82]. This is a full blown first-order dynamic logic of structured programs (commands) extended to handle procedure and function declarations, array and indexed data types, and parameter passing mechanisms. Goldblatt also shows his systems to be sound and complete. Note, however, mixed with Wand's generally positive review of [GOL82] the following criticism: "... the semantics of these constructs is unconvincing. Parameters, for example, are passed by global side-effect, with scoping and name conflicts handled via an ad hoc mechanism. It seems unlikely that a convincing semantics for these constructs can be developed without the introduction of explicit environments in the Scott-Strachey style. With this semantics, however, even the soundness of the rules becomes difficult." [WAN85;pp.855]

Segerberg's work

The modal logician Krista Segerberg, who incidentally was the first to announce a completeness result for propositional dynamic logic, has been successfully applying a variant of dynamic logic to actions in general. See especially [SEG80], [SEG82], [SEG85] and others. The latter of these papers modify somewhat the original "look" of dynamic logic and extend its substance.

Pratt's example #2

The examples of this section are from [PRA79c;pp.12-13].

Sagan

Source: *The Dragons of Eden: Speculations on the evolution of human intelligence* by Carl Sagan (Random House, New York (1977)). "It is because of [the] immense number of functionally different configurations of the human brain that no two humans, even identical twins raised together, can ever be really much alike. These enormous numbers may also explain something of the unpredictability of human behaviour and those moments when we surprise ourselves by what we do. Indeed, in the face of these numbers, the wonder is that there are any regularities at all in human behaviour. The answer must be that all possible brain states are by no means occupied; there must be an

enormous number of mental configurations that have never been entered or even glimpsed by any human being in the history of mankind. From this perspective, each human being is truly rare and different and the sanctity of individual human lives is a plausible ethical consequence." [pp.42]

logic programming

Some programming languages do support (or at least attempt to approximate in implementation) nondeterministic programming. The most notable exceptions to the conventional (i.e., procedural) variety of programming languages are the so-called logic programming languages such as Prolog. (Leon Sterling and Ehud Shapiro's book *The Art of Prolog: Advanced Programming Techniques* (MIT Press, Cambridge, Mass. (1986)) dedicates the whole of chapter 14 (pp.205-237) to nondeterministic programming - "... in addition to being a powerful theoretical concept, nondeterminism is also useful for defining and implementing algorithms. This chapter shows how, by thinking nondeterministically, one can construct concise and efficient programs." pp.206)

soft determinism

Even if all events are determined the underlying complexity is inevitably too great to allow us to predict even the most accessible them. Soft determinism is the view that, although all events are determined, predicting them is too complex. Basically, this view says that nondeterminism is practical even though it is not true (in much the same way as Newtonian physics is practical (modulo scale) but not true).

support for nondeterminism

The following have been pointed out by Harel and Pratt [H&P78;pp.203]. (Quotes and examples given in this and the next paragraph are from [PRA79c;pp.19-20].)

DPDL

Deterministic propositional dynamic logic (or DPDL) is investigated in [BHP82] where completeness (due to J. Halpern) and exponential deterministic time decidability (of the validity problem) are shown. (These results are also given, independently, by M.K. Valiev in *Decision Complexity of Variants of Propositional Dynamic Logic*, MFCS proc 1980, LNCS 88, pp.656-664). R. Parikh has shown that the problem of deciding validity has a lower bound that matches the upper bound of [BHP82] to within a polynomial [PAR81].

SDPDL

Structured or strict deterministic propositional dynamic logic (or SDPDL) is from [H&R81] where it is established that the decidability of the validity problem is polynomially space complete. In fact, SDPDL does not fall prey to the exponential lower bound of this problem for nondeterministic propositional dynamic logic with tests (see chapter three, section 3.5 - the techniques used by [H&R81] are similar to those of our chapter three (i.e., modified Fischer-Ladner closures and filtrations)). [H&R81] also demonstrate that SDPDL lacks the expressive power of its nondeterministic counterpart.

CHAPTER TWO

LOGICAL PRELIMINARIES

In this section the formal syntax that will be used throughout this thesis is presented. A flexible definition of the language of dynamic logics is given and a series of languages are defined. From these languages, several propositional logics based on the well known Segerberg axioms are introduced, and preliminary theorems are established which will be used in the succeeding chapters. The central results of this thesis, such as the completeness theorems established in chapter four, rely on the notion of maximality and Lindenbaum's lemma. These are discussed in the last section of this chapter.

2.1 The Language of Elementary Propositional Dynamic Logic

All propositional dynamic languages determine two categories of expression - formulae and programs. All the well formed expressions of a dynamic language are determined by four mutually exclusive sets:

- (1) Σ_0 - an infinite set of atomic formulae (also known as propositional letters, propositional symbols, etc.),
- (2) Π_0 - a set (finite or infinite) of basic programs (also known as action or program letters, action or program symbols, etc.),
- (3) **OpForm** - the set of operators which take their arguments to compound formulae,

- (4) **OpProg** - the set of operators which take their arguments to compound programs.

A propositional dynamic language is a four-tuple $(\Sigma_0, \Pi_0, \mathbf{OpForm}, \mathbf{OpProg})$.

Elements of **OpForm** and **OpProg** are operators that take formulae (atomic or compound) and/or programs (basic or compound) as arguments. For convenience, an operators' arguments are always arranged so that those referring to programs (if any) precede those arguments which refer to formulae (if any). Operators are also assigned to a "category" in the following fashion. A derived category is an ordered pair (n, m) of integers. If $\bullet \in \mathbf{OpForm} \cup \mathbf{OpProg}$ takes on n programs and m formulae as arguments then the category of $\bullet$ is the derived category (n, m) . In addition, $\bullet$ is called an $(n+m)$ -ary operator. If $n+m=0$ then $\bullet$ is referred to as a constant. If $n+m=1$ we refer to $\bullet$ as a unary operator. If $n+m=2$ we refer to $\bullet$ as a binary operator.

A characteristic of all dynamic languages is the presence of the Kripke operator. The Kripke operator is the binary operator $\diamond \in \mathbf{OpForm}$ of category $(1, 1)$, which is reminiscent to scalar multiplication in vector spaces or modules. That is, $\diamond$ acts in much the same way as scalar multiplication, producing a compound formula from a program and a formula. **OpForm** is always assumed to contain the operator $\diamond$. Most treatments of dynamic languages in the literature prefer to replace $\diamond$ with a number of unary modal operators, written $\langle a \rangle$, where a is a program. In these formalizations of dynamic languages the number of these modal operators is the same as the number of programs. Our approach is preferred for its brevity.

From $\Sigma_0, \Pi_0, \mathbf{OpForm}$ and **OpProg** two mutually exclusive sets of expressions of dynamic languages are defined. Σ is the set of formulae and Π is the set of programs defined by simultaneous induction:

- (a) $\Sigma_0 \subseteq \Sigma$; $\Pi_0 \subseteq \Pi$
- (b) Let $\bullet$ be an operator of category (n, m) , $a_1, \dots, a_n$ be programs and $p_1, \dots, p_m$ be formulae:

If $\bullet \in \text{OpForm}$ then $\bullet(a_1, \dots, a_n, p_1, \dots, p_m) \in \Sigma$;

If $\bullet \in \text{OpProg}$ then $\bullet(a_1, \dots, a_n, p_1, \dots, p_m) \in \Pi$.

Σ is sometimes called the set of *well-formed formulae*. In the same vein, Π is called the set of *well-formed programs*. Taken together Σ and Π are called the *well-formed expressions* of the propositional dynamic language $(\Sigma_0, \Pi_0, \text{OpForm}, \text{OpProg})$. Without loss of specification, we will sometimes treat a language as being synonymous with the set of its well-formed expressions.

We reserve lower case roman letters (with or without subscripts) from a to h for programs, lower case roman letters (with or without subscripts) from i to o for integers and lower case roman letters (with or without subscripts) from p to t for formulae. The lower case roman letters from u to z will be allocated later on.

Since we are dealing, for the most part, with only unary and binary operators we will employ a more conventional notation for compound expressions: If $\bullet$ is a unary operator then write $(\bullet v)$ (and sometimes $(v^\bullet)$) instead of $\bullet(v)$. If $\bullet$ is a binary operator then write $(v \bullet u)$ instead of $\bullet(v, u)$. The $\diamond$ operator, however, will take the form $(\langle a \rangle p)$ instead of $\diamond(a, p)$ or $(a \diamond p)$. This notation is also more common in the literature.

Throughout this thesis **OpForm** will always be a set of operators from the set of all well-formed expressions into Σ containing the basic (fixed) set of modal connectives, $\text{OpForm}(\text{modal}) = \{\perp, \vee, \neg, \diamond\}$ (i.e., $\text{OpForm} \supseteq \text{OpForm}(\text{modal})$). $\perp$ is the 0-ary operator or constant called *falsum*, $\vee$ is the binary alternation operator of category (0,2), $\neg$ is the unary negation operator of category (0,1) and $\diamond$ is the binary operation of "applying a program to a proposition" or of "a program enabling a proposition" which is, as already stated, of category (1,1).

Naturally, the following well known abbreviations are used (the notation $\stackrel{\text{def}}{=}$ is used to indicate a permanent syntactical convenience):

$$(p \ \&\ \neg q) \stackrel{\text{def}}{=} (\neg((\neg p) \vee (\neg q))),$$

$$\begin{aligned}
 (p \rightarrow q) &=_{\text{def}} ((\neg p) \vee q), \\
 (p \equiv q) &=_{\text{def}} ((p \rightarrow q) \& (q \rightarrow p)), \\
 ([a]p) &=_{\text{def}} (\neg \langle a \rangle (\neg p)), \\
 \top &=_{\text{def}} (\neg \perp), \\
 \langle a \rangle^0 p &=_{\text{def}} p, \\
 \langle a \rangle^{n+1} p &=_{\text{def}} (\langle a \rangle (\langle a \rangle^n p)).
 \end{aligned}$$

Parentheses are added or omitted as clarity demands, and, in general, the outermost parentheses are always dropped. We regard the symbols $\neg$ and $\vee$, $\langle a \rangle$ and $[a]$ as binding more strongly than $\&$ and $\rightarrow$, which in turn bind more strongly than $\equiv$. The notations $\langle a \rangle$ and $[a]$ are called *modalities*.

Let $\text{SUBF}(p)$ denote the set of *subformulae* of a given formula p , that is, $\text{SUBF}(p)$ is the smallest set satisfying the following conditions:

- a) $p \in \text{SUBF}(p)$,
- b) if $\neg q \in \text{SUBF}(p)$ then $q \in \text{SUBF}(p)$,
- c) if $q \vee r \in \text{SUBF}(p)$ then both $q, r \in \text{SUBF}(p)$,
- d) if $\langle a \rangle q \in \text{SUBF}(p)$ then $q \in \text{SUBF}(p)$.

OpProg shall always include $\text{OpProg}(\text{regular}) = \{ \lambda, ;, +, * \}$ where λ is a constant (i.e., category (0,0)) denoting the identity program; $;$ is the binary concatenation operator of category (2,0), $+$ is binary operator of category (2,0) that denotes nondeterministic choice and $*$ is the unary iteration operator of category (1,0). As is our policy, we write $a+b$ instead of $+(a,b)$, $a;b$ in place of $;(a,b)$ and a^* for $*(a)$. $a;b$ is further abbreviated to ab . In the case where a single program, say a , is concatenated a finite number of times (i.e., $a;(a;(a; \dots (a;a) \dots))$ or some other equivalent form) it will be convenient to use the following convention:

$$\begin{aligned}
 a^0 &=_{\text{def}} \lambda, \\
 a^{n+1} &=_{\text{def}} aa^n.
 \end{aligned}$$

We set $\mathcal{L}_{PDL} = (\Sigma_0, \Pi_0, \text{OpForm(modal)}, \text{OpProg(regular)})$. Thus, for $\mathcal{L}_{PDL}$, Π , the set of well-formed programs, is the set of regular expressions over Π_0 , which we denote Π_{PDL} . Also, let $\Sigma_{PDL} = \Sigma$ denote the set of well-formed formulae of $\mathcal{L}_{PDL}$.

2.2 Extensions to the Language of Elementary Propositional Dynamic Logic

In the course of this thesis we will have need to refer to several different versions of propositional dynamic logic. Any form of propositional dynamic logic will be called a PDL. (NOTE: acronym) In the very least, every PDL will have programs that are regular expressions over Π_0 (i.e., Π_{PDL}) and formulae that are from Σ_{PDL} .

Some popular extensions to Σ_{PDL} include expressions for test programs and the reversal of programs. The symbol $?$ denotes a test operator of category (0,1). The symbol c is used to denote the operator of performing a program in reverse (known as backtracking) and is of category (1,0). In order to extend $\mathcal{L}_{PDL}$ we set OpProg(regular,?) to OpProg(regular) supplemented by $?$, OpProg(regular,c) to OpProg(regular) supplemented by c and $\text{OpProg(regular,?,c)}$ to be OpProg(regular) supplemented with both $?$ and c . Thus, we have

$$\mathcal{L}_{PDL,?} = (\Sigma_0, \Pi_0, \text{OpForm(modal)}, \text{OpProg(regular,?)}),$$

$$\mathcal{L}_{PDL,c} = (\Sigma_0, \Pi_0, \text{OpForm(modal)}, \text{OpProg(regular,c)}),$$

$$\mathcal{L}_{PDL,?,c} = (\Sigma_0, \Pi_0, \text{OpForm(modal)}, \text{OpProg(regular,?,c)}),$$

The formula *repeat(a)* is used to indicate that the program a^* can enter into a non-halting computation (i.e., a^* can diverge). Thus *repeat* is an operator of category (1,0) that determines a formula from a given program. Set $\text{OpForm(modal,repeat)}$ be OpForm(modal) supplemented by *repeat*. Thus, we have

$$\mathcal{L}_{PDL,\text{repeat}} = (\Sigma_0, \Pi_0, \text{OpForm(modal,repeat)}, \text{OpProg(regular)}).$$

Likewise, let

$$\begin{aligned}\mathcal{L}_{\text{PDL},?,\text{repeat}} &= (\Sigma_0, \Pi_0, \text{OpForm}(\text{modal},\text{repeat}), \text{OpProg}(\text{regular},?)), \\ \mathcal{L}_{\text{PDL},c,\text{repeat}} &= (\Sigma_0, \Pi_0, \text{OpForm}(\text{modal},\text{repeat}), \text{OpProg}(\text{regular},c)), \\ \mathcal{L}_{\text{PDL},?,c,\text{repeat}} &= (\Sigma_0, \Pi_0, \text{OpForm}(\text{modal},\text{repeat}), \text{OpProg}(\text{regular},?,c)).\end{aligned}$$

For a language $\mathcal{L}_{\text{subscript}}$, $\Pi_{\text{subscript}}$ and $\Sigma_{\text{subscript}}$ are used to denote its well-formed programs and well-formed formulae, respectively. Thus, for example, we have the following proper inclusions:

$$\begin{aligned}\Pi_{\text{PDL}} &\subset \Pi_{\text{PDL},?} \subset \Pi_{\text{PDL},?,c} \subset \Pi_{\text{PDL},?,c,\text{repeat}} \\ \Sigma_{\text{PDL}} &\subset \Sigma_{\text{PDL},?} \subset \Sigma_{\text{PDL},?,c} \subset \Sigma_{\text{PDL},?,c,\text{repeat}} \\ \mathcal{L}_{\text{PDL}} &\subset \mathcal{L}_{\text{PDL},?} \subset \mathcal{L}_{\text{PDL},?,c} \subset \mathcal{L}_{\text{PDL},?,c,\text{repeat}}\end{aligned}$$

Unless otherwise indicated we will assume that Σ , Π and $\mathcal{L}$ are subsets of $\Sigma_{\text{PDL},?,c,\text{repeat}}$, $\Pi_{\text{PDL},?,c,\text{repeat}}$ and $\mathcal{L}_{\text{PDL},?,c,\text{repeat}}$ respectively.

The reader should be warned that these are not trivial extensions of $\mathcal{L}_{\text{PDL}}$ - they are more than mere "syntactical sugar" (to borrow a popular term from the programming language milieu). Chapter five is dedicated to showing that each additional operator adds to the real expressive power of preceding PDLs.

2.3 Axioms and Rules

We are now in a position to formulate the axiomatic basis for propositional dynamic logics. In this section to introduce rules and axiom schema, which, beside being clearly "valid" and "natural" ways for arguing about programs, lead to desirous completeness results (of chapter four). We will not undertake a justification of these rules or axioms beyond the fact that they appear to coincide with our empirical evidence, not only of programs, but also of actions in general. To describe the axioms

introduced, we use the terms *valid* (i.e., meaning *always true*), *true* (i.e., meaning *sometimes true*) and *satisfied* (i.e., meaning *interpreted*) loosely, but in accordance with our pre-formal intuitions. The formal treatment of these terms is given in chapter three.

Of course, since propositional dynamic logics are extensions of classical propositional logic (CPL), we include in the underlying logic all propositional tautologies and the rule of modus ponens: A set of formulae L in the language $\mathcal{L}$ is a propositional logic if it satisfies the following two conditions:

- (PT) L contains all propositional tautologies (including the constant symbol $\top$),
- (MP) L is closed under modus ponens, i.e., if $p, p \rightarrow q \in L$ then $q \in L$,

Taken together we shall refer to PT and MP by PC. Classical propositional logic, or CPL for short, is the smallest propositional logic.

Let L be any propositional logic in the language $\mathcal{L}$ (as defined in the last paragraph). We say that a formula p is deducible in L , or that p is provable in L or that p is a theorem of L , and write $\vdash_L p$ whenever $p \in L$. We extend this definition to say that a formula p is deducible in L from a set Γ of formulae, in symbols $\Gamma \vdash_L p$, if there is a finite number $n \geq 0$ of formulae $q_1, \dots, q_n \in \Gamma$ such that $\vdash_L q_1 \& \dots \& q_n \rightarrow p$. When $n=0$ then this condition reduces to $\vdash_L p$. (Whenever there is no chance of ambiguity arising we sometimes omit subscripts from the $\vdash_L$ notation.)

Let Γ be a set of formulae and L be a propositional logic. If $\Gamma \vdash_L \perp$ then Γ is said to be L -inconsistent. Otherwise Γ is said to be L -consistent. We say that L is inconsistent (consistent) whenever $\emptyset$ is L -inconsistent (L -consistent). Thus, L is inconsistent if and only if $\vdash_L \perp$ and L is consistent if and only if not $\vdash_L \perp$. When Γ is a singleton set of formulae, say $\{p\}$, then we say that p is L -consistent or L -inconsistent just when Γ is.

2.3.1 Deduction theorems: Let L be any logic and $\Gamma, \Upsilon \subseteq \Sigma$ be sets of formulae, then

- (a) if $p \in \Gamma$ then $\Gamma \vdash_L p$
- (b) if $\Gamma \vdash_L q$ and $\{q\} \vdash_L p$ then $\Gamma \vdash_L p$
- (c) if $\Gamma \vdash_L p$ and $\Gamma \subseteq \Upsilon$ then $\Upsilon \vdash_L p$
- (d) $\Gamma \vdash_L p$ iff there is a finite subset $\Upsilon \subseteq \Gamma$ such that $\Upsilon \vdash_L p$
- (e) $\Gamma \vdash_L p \rightarrow q$ iff $\Gamma \cup \{p\} \vdash_L q$.

Proof:

- (a) Since $p \in \Gamma$ and $\vdash p \rightarrow p$ (by PC) then $\Gamma \vdash p$.
- (b) If $\{q\} \vdash p$ then either $\vdash p$ (in which case $\Gamma \vdash p$ automatically) or $\vdash q \rightarrow p$. If the latter, considering $\Gamma \vdash q$, then there are $q_1, \dots, q_n \in \Gamma$ such that $\vdash q_1 \& \dots \& q_n \rightarrow q$ so that, via PC, $\vdash q_1 \& \dots \& q_n \rightarrow p$. Therefore $\Gamma \vdash p$.
- (c) and (d) follow directly from the definition of $\Gamma \vdash p$.
- (e) $\Gamma \vdash p \rightarrow q$ iff there exists a finite number of formulae of Γ say $r_1, \dots, r_n$ such that $\vdash r_1 \& \dots \& r_n \rightarrow (p \rightarrow q)$. Now by PC, this is tantamount to $\vdash r_1 \& \dots \& r_n \& p \rightarrow q$, which in turn is equivalent to $\Gamma \cup \{p\} \vdash q$. ■

2.3.2 Syntactic compactness theorem: Γ is L -consistent if and only if every finite subset of Γ is L -consistent.

Proof: Suppose Γ has an L -inconsistent finite subset Γ' . If $\Gamma' \vdash p$ then $\Gamma \vdash p$ (2.3.1c), therefore Γ must be L -inconsistent also. On the other hand, suppose $\Gamma \vdash \perp$ then there is a finite number of formulae say $q_1, \dots, q_n$ such that $\vdash q_1 \& \dots \& q_n \rightarrow \perp$. Thus $\{q_0, q_1, \dots, q_n\}$ is also L -inconsistent. ■

2.3.3 Theorem: If Γ is L -consistent and $\Upsilon \subseteq \Gamma$ then Υ is also L -consistent.

Proof: $\Upsilon \vdash p$ implies $\Gamma \vdash p$, so that if Υ is L -inconsistent then so is Γ (2.3.1c). ■

By the term normal (multi-modal) propositional logic we mean a propositional logic N that also satisfies two additional conditions:

(Ax.K) N contains every formula of the form

$$[a](p \rightarrow q) \rightarrow [a]p \rightarrow [a]q,$$

(RN) for every $a \in \Pi$, the logic N is closed under the rule of a-

necessitation, i.e., $\vdash_N p$ implies $\vdash_N [a]p$.

To justify the axiom schema Ax.K suppose that, after any execution of the program a , the expression $p \rightarrow q$ is true. From this, it is obviously legitimate to conclude that, if p is always true upon the termination of a , then likewise, q shall always be true. The validity of RN is also transparent - if p is a valid condition (i.e., true under all circumstances), the execution of any program shall not invalidate it (so long as the starting condition is one recognized by a).

Let PKL be the smallest normal propositional logic (K is historically the symbol reserved for the smallest normal modal propositional logic, named after Kripke). PKL is all too clearly an inadequate system for arguing about any extensive concept of programs. It doesn't really amount to anything more than a collection of normal modal logics. No significant program manipulation is provided for. We now introduce some axiom schemata that have been proposed for this purpose.

The execution of a program which is the nondeterministic choice of two programs, say $a+b$, is simply the execution of a or the execution of b . Thus, the following axiom schema is valid:

$$(Ax.+) \quad \langle a+b \rangle p \equiv \langle a \rangle p \vee \langle b \rangle p .$$

As a reflection of the intended interpretation of $+$, Ax.+ is often called the union axiom in the literature.

The composition of two programs, say ab , reflects the action of "doing a followed by b ". Naturally, given that $\langle ab \rangle p$ (i.e., that p is sometimes brought about to be true by the composite program ab) we may deduce that $\langle b \rangle p$ is sometimes made true by the execution of the program a , and vice versa. Thus, the following formula is a valid statement about programs:

$$(Ax. ;) \quad \langle ab \rangle p \equiv \langle a \rangle \langle b \rangle p .$$

In the literature $Ax.;$ is often referred to as the composition axiom.

The principle of iteration involves the execution of a program an undetermined number of times - possibly zero. Iteration is captured by the following axiom schemata:

$$(Ax.*) \quad p \vee \langle a \rangle \langle a^* \rangle p \rightarrow \langle a^* \rangle p .$$

$$(Ax.Ind) \quad \langle a^* \rangle p \rightarrow p \vee \langle a^* \rangle (\neg p \& \langle a \rangle p) .$$

The dual of $Ax.*$ is the formula

$$(IT) \quad [a^*]p \rightarrow p \& [a][a^*]p .$$

Thus, whenever an arbitrary number of executions of a program a always brings about the truth of p , we may deduce that the number of executions of a has no effect on p . Accordingly, schema $Ax.*/IT$ allows us to extend or release part of the modality on condition p (see, in particular, theorems 2.5.1, 2.5.2 and 2.5.4 below which bundled together are equivalent to $Ax.*/IT$). $Ax.Ind$ is called the induction axiom, better known by its dual form:

$$(IN) \quad p \& [a^*](p \rightarrow [a]p) \rightarrow [a^*]p .$$

Therefore, $Ax.Ind$ is a simple induction axiom, allowing us to infer absolute invariance of a program a on proposition p from induction conditions based on the number of executions of a . (NOTE: 1st-order decidability)

The axiom schema for the identity program is as follows:

$$(Ax.Id) \quad \langle \lambda \rangle p \equiv p .$$

That is to say, λ is the program which does nothing but halt - it is a program which will not affect the truth status of any proposition. Therefore λ is included as a purely

syntactic convenience. (This is especially true in the presence of tests, as shown by theorem 2.5.9 below.)

All together PC, RN, Ax.K, Ax.+, Ax.~, Ax.* and Ax.Ind (or some equivalent set of axioms) are known as the *Segerberg axioms (and rules)*. (In this thesis we expand this grouping to include Ax.id.) Ax.+ through Ax.* are due to Pratt [PRA76]. Ax.Ind is due to Segerberg [SEG77]. (NOTE: induction axiom) A normal propositional logic that contains, at least, all of the Segerberg axioms and rules is called a *propositional dynamic logic* (or a PDL).

Let P (in deference to Pratt) denote the smallest PDL. That is, P is the smallest logic which satisfies the Segerberg conditions and whose expressions are from $\mathcal{L}_{PDL}$. P is the most elementary dynamic logic that we will be studying in this thesis (sometimes it is called test free PDL). Now we define some extensions to P .

A rule (which will be of particular interest to us when we take up completeness in chapter four) is usually associated with deontic logic. Consider the rule,

$$(RP) \quad \vdash_L p \text{ implies } \vdash_L \langle a \rangle p.$$

where L is a PDL. (NOTE: deontic logic) Using this rule we may infer from valid p , of any program a , the possibility that a will terminate (regardless of the condition in which a starts). However, RP is definitely not valid. RP can only be used when all programs are *serial*, which means that they must recognize every possible starting condition. In the formal semantics that we set out in chapter three, it is possible for programs not to be defined from many possible starting conditions. Let D denote the smallest PDL which contains the logic P and is closed under RP (and whose expressions are from $\mathcal{L}_{PDL}$).

Our test axiom shall be:

$$(Ax.?) \quad \langle p? \rangle q \equiv p \& q.$$

Thus, if the program $p?$ should terminate (meaning that p is true) leaving the sentence $\langle p? \rangle q$ true then q must also be true, and conversely, given q and p then $\langle p? \rangle q$ will certainly follow. The reader will have noticed that with $Ax.?$ we could eliminate the standard **OpForm**(modal) operator $\vee$ (since $\&$ is defined in terms of $\vee$ and $\neg$). That is to say, $\vee$ does not add to the expressive power of PDLs with tests. We could define the form $p\vee q$ to be an abbreviation of $[\neg p?]q$ (i.e., $p\vee q =_{\text{def}} [\neg p?]q$). However, we cannot as simply remove tests - they may be bound within an $*$ operation. Indeed, as we shall find in chapter five (section 5.1), tests do add to the expressive power of PDLs.

The axiom for the converse operator is:

$$(Ax.c) \quad p \rightarrow [a]\langle a^c \rangle p \ \& \ [a^c]\langle a \rangle p.$$

That is, given a program executed from a starting state, say w , which satisfies proposition p , there is at least one reverse execution of the same program which will take us to a condition that once again satisfies p (namely the reverse execution which takes us back to condition w , although there may be others). Note that, so long as the programs we consider are generally nondeterministic we cannot have $p \rightarrow [a][a^c]p$. Even when program a is deterministic (i.e., starting from any initial condition, a has a single terminating state) its converse a^c may not be deterministic.

There are two axioms for the repeat operator:

$$(Ax.r1) \quad \text{repeat}(a) \equiv \langle a \rangle \text{repeat}(a),$$

$$(Ax.r2) \quad [a^*](p \rightarrow \langle a \rangle p) \rightarrow (p \rightarrow \text{repeat}(a)).$$

The first of these is obvious. Assume that, the program a^* can run forever (i.e., executing a repeatedly an infinite number of times). Then clearly, there must be some execution of a after which a may be executed over and over again without ever terminating. Conversely, if after some execution of a , a^* can run forever then a^* may run forever. The second of these axioms of the repeat operator specifies a sufficient

condition for a proposition p to imply the that a program a may be executed consecutively, infinitely often: after any finite number of executions of a , p must imply that it is possible to execute a again in such a way that p will remain true.

Set $P?$ to denote the smallest logic that satisfies the Segerberg axioms supplemented by $Ax.?$ and whose expressions are from $\mathcal{L}_{PDL,?}$. ($P?$ is nearly always considered to be the standard PDL.) P_c denotes the smallest logic that satisfies the Segerberg axioms supplemented by $Ax.c$ and whose expressions are from $\mathcal{L}_{PDL,c}$, and, similarly, P_r denotes the smallest logic that satisfies the Segerberg axioms supplemented by $Ax.r1$ and $Ax.r2$ and whose expressions are from $\mathcal{L}_{PDL,repeat}$. $P?c$ is the smallest logic that satisfies the Segerberg axioms supplemented by $Ax.?$ and $Ax.c$ and whose expressions are from $\mathcal{L}_{PDL,?,c}$. Continuing in this way, also define $P?r$, P_{cr} and $P?cr$ by supplementing P with the indicated axioms and extending the language of expression appropriately. Likewise define $D?$, D_c , $D?c$ and $D?cr$, i.e., those logics which result from adding some of the axioms $Ax.?$, $Ax.c$, $Ax.r1$ and $Ax.r2$ to the logic D and extending the set of expressions appropriately.

2.4 Immediate Results: Normal Propositional Modal Logics

The following well known results are theorem schema of all normal propositional logics, including PKL:

2.4.1 **Theorem:** a) $\vdash \langle a \rangle \perp \equiv \perp$

b) $\vdash [a] \top \equiv \top$

Proof: a) $\vdash \perp \rightarrow \langle a \rangle \perp$ (by PT). Also $\vdash [a] \top$ (by RN) so $\vdash \top \rightarrow [a] \top$ (by PT) or equivalently $\vdash \langle a \rangle \perp \rightarrow \perp$. Now, combining these results by PC, gives $\vdash \langle a \rangle \perp \equiv \perp$. b) is the dual of a). ■

In the literature of modal logic the following rules of inference are usually referred to as RM, RB (Becker's rule) and RN $\circ$ respectively. RM is sometimes called the rule of distributivity (e.g., [HAR84;pp.513]).

2.4.2 Theorem: For all $a \in \Pi_{PDL}$,

(RM) $\vdash p \rightarrow q$ implies $\vdash [a]p \rightarrow [a]q$,

(RB) $\vdash p \rightarrow q$ implies $\vdash \langle a \rangle p \rightarrow \langle a \rangle q$,

(RN $\diamond$) $\vdash \neg p$ implies $\vdash \neg \langle a \rangle p$.

Proof:

For RM: from $\vdash p \rightarrow q$, $\vdash [a](p \rightarrow q)$ by RN. The result follows by Ax.K and MP.

For RB: Given $\vdash p \rightarrow q$ then $\vdash \neg q \rightarrow \neg p$ by PC, $[a]\neg q \rightarrow [a]\neg p$ by RM and $\vdash \langle a \rangle p \rightarrow \langle a \rangle q$ by PC.

RN $\diamond$ is just the dual of RN. ■

2.4.3 Theorem: a) $\vdash [a](p \& q) \equiv [a]p \& [a]q$

b) $\vdash \langle a \rangle (p \vee q) \equiv \langle a \rangle p \vee \langle a \rangle q$

Proof: a) $p \& q \rightarrow p$ (PT) therefore $[a](p \& q) \rightarrow [a]p$ (by RM). Similarly $[a](p \& q) \rightarrow [a]q$. The combination of these two results by PC yields the desired result. Conversely, $p \rightarrow (q \rightarrow (p \& q))$ (by PT) so $[a]p \rightarrow [a](q \rightarrow p \& q)$ (by RM), therefore $[a]p \& [a]q \rightarrow [a](p \& q)$ (by Ax.K and PC). b) is the dual of a). ■

In the literature on dynamic algebra, $\diamond$ is often called "scalar multiplication" because Ax.+, Ax.; and 2.4.3 are reminiscent of the axioms of scalar multiplication for vector spaces or modules.

2.4.4 Theorem: a) $\vdash (p \vee \neg p) \equiv \top$

b) $\vdash (p \& \neg p) \equiv \perp$

Proof: By PC. ■

2.4.5 Theorem: a) $\vdash \langle a \rangle (p \& q) \rightarrow \langle a \rangle p \& \langle a \rangle q$

b) $\vdash [a]p \vee [a]q \rightarrow [a](p \vee q)$

Proof: a) $p \& q \rightarrow p$ and $p \& q \rightarrow q$ (PT). The required result follows by RB and PC. b) is the dual of a). ■

2.5 Immediate Results: Propositional Dynamic Logics

The theorems of this section are outside the realm of classical modal logic because their proofs employ axioms over and above those of PKL.

Theorems concerning "regular programs"

Unless otherwise indicated, the results of this section are theorems for any PDL and P in particular. That is to say, the following theorems 2.5.1 to 2.5.7 only employ the Segerberg axioms in their proofs.

2.5.1 Theorem: a) $\vdash [a^*]p \rightarrow p$

b) $\vdash p \rightarrow \langle a^* \rangle p$ (NOTE: $\langle a^* \rangle$ deontic modality)

Proof: a) From IT and PC. b) is the dual of a). ■

2.5.2 Theorem: a) $\vdash [a^*]p \rightarrow [a][a^*]p$

b) $\vdash \langle a \rangle \langle a^* \rangle p \rightarrow \langle a^* \rangle p$

Proof: a) From IT and PC. b) is the dual of a). ■

2.5.3 Theorem: a) $\vdash [a^*]p \rightarrow [a]p$

b) $\vdash \langle a \rangle p \rightarrow \langle a^* \rangle p$

Proof: From 2.5.2, 2.5.1 and PC. ■

2.5.4 Theorem: a) $\vdash [a^*]p \rightarrow [a^*][a^*]p$ (NOTE: $[a^*]$ S4 modality)

b) $\vdash \langle a^* \rangle \langle a^* \rangle p \rightarrow \langle a^* \rangle p$

Proof: a) $\vdash [a^*]([a^*]p \rightarrow [a][a^*]p)$ from 2.5.2 and RN. The result follows from IN and MP. b) is the dual of a). ■

The following theorem gives the rule of invariance, or just RI. It turns out that RI is actually equivalent to IN/Ax.Ind in PDLs.

2.5.5 Theorem: (RI) $\vdash p \rightarrow [a]p$ implies $\vdash p \rightarrow [a^*]p$

$(RI_0) \vdash \langle a \rangle p \rightarrow p$ implies $\vdash \langle a^* \rangle p \rightarrow p$

Proof: For RI , suppose $\vdash p \rightarrow \{a\}p$, so that $\vdash [a^*](p \rightarrow \{a\}p)$. By IN and MP $\vdash \neg p \vee [a^*]p$, or $\vdash p \rightarrow [a^*]p$. RI_0 is the dual of RI . ■

2.5.6 Theorem: $\vdash [a+b]p \equiv [a]p \& [b]p$.

Proof: Dual of $Ax.+$. ■

2.5.7 Theorem: $\vdash [ab]p \equiv [a][b]p$.

Proof: Dual of $Ax.;$. ■

Theorems concerning tests

The following are theorems that require $Ax.?$ in their proofs. Let L be a PDL containing $P?$.

2.5.8 Theorem: $\vdash_L [p?]q \equiv p \rightarrow q$.

Proof: This is the dual of $Ax.?$. ■

Thus, "at termination the program $p?$ will always leave q true" is synonymous with q being implicit from p .

2.5.9 Theorem: $\vdash_L \langle \lambda \rangle p \equiv \langle T? \rangle p$.

Proof: By $Ax.id$ and $Ax.?$. ■

Actually, in $P?$ and its extensions we could eliminate the identity program entirely by making $T?$ synonymous with λ .

Theorems concerning converse

The following are theorems that employ $Ax.c$ in their proofs. Let L be a PDL containing Pc . Note that theorems 2.5.10 and 2.5.11 are equivalent in strength to $Ax.c$.

2.5.10 Theorem: $\vdash_L \langle a \rangle [a^c] p \rightarrow p$.

Proof: Dual of $\vdash_L p \rightarrow [a] \langle a^c \rangle p$, which is immediate from Ax.c ■

2.5.11 Theorem: $\vdash_L \langle a^c \rangle [a] p \rightarrow p$.

Proof: Dual of $\vdash_L p \rightarrow [a^c] \langle a \rangle p$, which is immediate from Ax.c ■

Recall, from the introduction, that the formula $p \rightarrow [a]q$ ($p \{a\} q$ in the symbolism of Floyd-Hoare logic) asserts that if the precondition p is true prior to the execution of a then, if a terminates, the postcondition q will be true. This is the so called partial correctness assertion or PCA of a with respect to precondition p and postcondition q . The following theorem gives an interesting duality with PCA's, employing converse:

2.5.12 Theorem: a) $\vdash_L p \rightarrow [a]q$ if and only if $\vdash_L \langle a^c \rangle p \rightarrow q$.

b) $\vdash_L \langle a \rangle p \rightarrow q$ if and only if $\vdash_L p \rightarrow [a^c]q$.

Proof: a) If $\vdash p \rightarrow [a]q$ then, by RB, also $\vdash \langle a^c \rangle p \rightarrow \langle a^c \rangle [a]q$. The result is immediate by 2.5.11 and the transitivity of $\rightarrow$ (PC). Conversely, if $\vdash \langle a^c \rangle p \rightarrow q$ then $\vdash [a] \langle a^c \rangle p \rightarrow [a]q$ by RM. Ax.c and PC yield $\vdash p \rightarrow [a]q$. b) is the dual of a). ■

The motivation for the PCA $p \rightarrow [a]q$ is as follows. If program a is started while p is true, then, in terms of what a is advertised to do - as expressed by the postcondition q - we ask what is true when a terminates. This is called forward reasoning because it proceeds from start to end. Obviously, the weakest liberal precondition of the PCA $p \rightarrow [a]q$ is $p = [a]q$. Backward reasoning, so called because it starts with assumptions about a 's termination, asks what precondition will guarantee the termination of a (if indeed the termination of a is possible) while bringing about the truth of q in the process. According to 2.5.12a, $q = \langle a^c \rangle p$ is the strongest postcondition.

Theorems 2.5.13 to 2.5.17 (and 2.5.22 to 2.5.26 when *repeat* is introduced) demonstrate that converse is very well behaved - and show that, without loss of expressive power, we could have restricted the converse operator to atomic programs, Π_0 . This fact is essential for the proof that $P?$ is more expressive than P_c (chapter five, theorem 5.1.7).

2.5.13 Theorem: a) $\vdash_L \langle (ab)^c \rangle p \equiv \langle b^c \rangle \langle a^c \rangle p$.

b) $\vdash_L [(ab)^c]p \equiv [b^c][a^c]p$.

Proof: a) Starting from the tautology $\vdash \langle b^c \rangle \langle a^c \rangle p \rightarrow \langle b^c \rangle \langle a^c \rangle p$, by 2.5.12a (twice) we have $\vdash p \rightarrow [a][b] \langle b^c \rangle \langle a^c \rangle p$, and so by 2.5.7, $\vdash p \rightarrow [ab] \langle b^c \rangle \langle a^c \rangle p$. By 2.5.12a, $\vdash \langle (ab)^c \rangle p \rightarrow \langle b^c \rangle \langle a^c \rangle p$.

Conversely, we have $\vdash p \rightarrow [ab] \langle (ab)^c \rangle p$ (Ax.c), and so by 2.5.7, $\vdash p \rightarrow [a][b] \langle (ab)^c \rangle p$. By 2.5.12a (twice) $\vdash \langle b^c \rangle \langle a^c \rangle p \rightarrow \langle (ab)^c \rangle p$. The result is immediate from Ax.:. b) is the dual of a). ■

2.5.14 Theorem: a) $\vdash_L \langle (a+b)^c \rangle p \equiv \langle a^c \rangle p \& \langle b^c \rangle p$.

b) $\vdash_L [(a+b)^c]p \equiv [a^c]p \& [b^c]p$.

Proof: According to Ax.c, we have $\vdash p \rightarrow [a+b] \langle (a+b)^c \rangle p$. Thus, by 2.5.6 and PC, $\vdash p \rightarrow [a] \langle (a+b)^c \rangle p$ and $\vdash p \rightarrow [b] \langle (a+b)^c \rangle p$. By 2.5.12a (on both of these formulae) we obtain $\vdash \langle a^c \rangle p \rightarrow \langle (a+b)^c \rangle p$ and $\vdash \langle b^c \rangle p \rightarrow \langle (a+b)^c \rangle p$ respectively. Therefore, by PC, $\vdash \langle a^c \rangle p \& \langle b^c \rangle p \rightarrow \langle (a+b)^c \rangle p$.

Conversely, $p \rightarrow [a] \langle a^c \rangle p$ (Ax.c), therefore $p \rightarrow [a] \langle a^c \rangle p \vee [b] \langle a^c \rangle p$ (by PC) and, so by Ax.+, $p \rightarrow [a+b] \langle a^c \rangle p$, and $\langle (a+b)^c \rangle p \rightarrow \langle a^c \rangle p$ by 2.5.12a. Similarly, $\langle (a+b)^c \rangle p \rightarrow \langle b^c \rangle p$. Combining these results (by PC) we obtain $\langle (a+b)^c \rangle p \rightarrow \langle a^c \rangle p \& \langle b^c \rangle p$. b) is the dual of a). ■

Since they do not add to clarity we omit parentheses to separate consecutive unary operators. For example it should be clear that $[a^{*c}]$ and $[a^{cc}]$ are short for the modalities $[(a^*)^c]$ and $[(a^c)^c]$ respectively - i.e., the argument "closest" to the operator is applied first.

2.5.15 Theorem: a) $\vdash_L \langle a^{cc} \rangle p \equiv \langle a \rangle p$.

b) $\vdash_L [a^{cc}]p \equiv [a]p$.

Proof: a) $p \rightarrow [a^c] \langle a \rangle p$ (Ax.c) therefore, by 2.5.12b, $\langle a^{cc} \rangle p \rightarrow \langle a \rangle p$.

Conversely, $p \rightarrow [a^c] \langle a^{cc} \rangle p$ (Ax.c) therefore, by 2.5.12b, $\langle a \rangle p \rightarrow \langle a^{cc} \rangle p$. b) is the dual of a). ■

2.5.16 Theorem: a) $\vdash_L \langle a^{*c} \rangle p \equiv \langle a^{*c} \rangle p$.

$$b) \vdash_L [a^{*c}]p \equiv [a^{c*}]p.$$

Proof: b) According to 2.5.2a, $[a^{c*}]p \rightarrow [a^c][a^{c*}]p$, so that by 2.5.12a, $\langle a \rangle [a^{c*}]p \rightarrow [a^{c*}]p$ also. Thus, by RI $\diamond$, $\langle a^* \rangle [a^{c*}]p \rightarrow [a^{c*}]p$. Now, via 2.5.1a, we have $\langle a^* \rangle [a^{c*}] \rightarrow p$. Therefore, by RM, $[a^{c*}] \langle a^* \rangle [a^{c*}]p \rightarrow [a^{c*}]p$. Also, $[a^{c*}]p \rightarrow [a^{c*}] \langle a^* \rangle [a^{c*}]p$ (Ax.c). By the transitivity of $\rightarrow$ in PC, $[a^{c*}]p \rightarrow [a^{c*}]p$. (Note that the dual of this result is

$$\langle a^{*c} \rangle p \rightarrow \langle a^{c*} \rangle p \quad (1)$$

which is used in the second part of this proof.)

Conversely, from Ax.c we have $[a^{cc*}]p \rightarrow [a^{c*}] \langle a^{cc*} \rangle [a^{cc*}]p$ and therefore, by (1) and PC, $[a^{cc*}]p \rightarrow [a^{c*}] \langle a^{cc*} \rangle [a^{cc*}]p$. And so, by 2.5.10,

$$[a^{cc*}]p \rightarrow [a^{c*}]p. \quad (2)$$

Now, from $\langle a^{cc} \rangle [a^{c*}]p \rightarrow \langle a \rangle [a^{c*}]p$ (2.5.15) we have, by RI $\diamond$, $\langle a^{cc*} \rangle [a^{c*}]p \rightarrow \langle a \rangle [a^{c*}]p$. Thus, according to 2.5.3b, $\langle a^{cc*} \rangle [a^{c*}]p \rightarrow \langle a^* \rangle [a^{c*}]p$ also. But since $\langle a^* \rangle [a^{c*}]p \rightarrow p$ (2.5.10) then $\langle a^{cc*} \rangle [a^{c*}]p \rightarrow p$ or $[a^{c*}]p \rightarrow [a^{cc*}]p$. Combining this result with (2) gives $[a^{c*}]p \equiv [a^{c*}]p$.

a) is the dual of b). ■

In addition to Ax.c, the following theorem requires Ax.?, i.e., L is a PDL containing P?c.

2.5.17 Theorem: a) $\vdash_L \langle p? \rangle q \equiv \langle (p?)^c \rangle q$,

$$b) \vdash_L [p?]q \equiv [(p?)^c]q.$$

Proof: a) To verify $\langle (p?)^c \rangle q \rightarrow \langle p? \rangle q$, reduce to a tautology by 2.5.12a and Ax.?. For the converse, start with $q \rightarrow [p?]\langle (p?)^c \rangle q$ (Ax.c). Then, by 2.5.8, $q \rightarrow (p \rightarrow \langle (p?)^c \rangle q)$, so that by PC, $q \& p \rightarrow \langle (p?)^c \rangle q$. Finally $\langle p? \rangle q \rightarrow \langle (p?)^c \rangle q$ by Ax._i?.

b) is the dual of a). ■

Theorems concerning infinite executions

The following are theorems that require Ax.r1 and Ax.r2 in their proofs. Let L be a PDL containing Pr.

2.5.18 Theorem: $\vdash_L \text{repeat}(a) \rightarrow \langle a \rangle \top$.

Proof: Now, by PT, $\text{repeat}(a) \rightarrow \top$. Then, by RB, $\langle a \rangle \text{repeat}(a) \rightarrow \langle a \rangle \top$. And so, according to Ax.r1 and by PC, $\text{repeat}(a) \rightarrow \langle a \rangle \top$. ■

2.5.19 Theorem: $\vdash_L \text{repeat}(a) \vee \text{repeat}(b) \equiv \text{repeat}(a+b)$.

Proof: Let $p = \text{repeat}(a) \vee \text{repeat}(b)$. From Ax.r1 and PC, it follows that $\text{repeat}(a) \rightarrow \langle a \rangle \text{repeat}(a) \vee \langle b \rangle \text{repeat}(a)$, so that $\text{repeat}(a) \rightarrow \langle a+b \rangle \text{repeat}(a)$ by Ax.+ . Likewise, $\text{repeat}(b) \rightarrow \langle a+b \rangle \text{repeat}(b)$. Therefore, by PC and 2.4.3b, $p \rightarrow \langle a+b \rangle p$. By RN, $[(a+b)^*](p \rightarrow \langle a+b \rangle p)$. Finally, by Ax.r2, $\text{repeat}(a) \vee \text{repeat}(b) \rightarrow \text{repeat}(a+b)$. Conversely, let $p = \text{repeat}(a+b)$. By Ax.r1, $p \rightarrow \langle a+b \rangle p$, so that $p \rightarrow \langle a \rangle p \vee \langle b \rangle p$ (by 2.4.3b). Thus, by PC, either $p \rightarrow \langle a \rangle p$ or $p \rightarrow \langle b \rangle p$, and, in turn, either $[a^*](p \rightarrow \langle a \rangle p)$ or $[b^*](p \rightarrow \langle b \rangle p)$ (by RN), respectively. Consequently, through Ax.r2 we have either $p \rightarrow \text{repeat}(a)$ or $p \rightarrow \text{repeat}(b)$. Combining these results by PC gives the desired result. ■

2.5.20 Theorem: $\vdash_L \text{repeat}(a^*)$.

Proof: We know by 2.5.1b that $\top \rightarrow \langle a^* \rangle \top$. So, by RN, we have $[a^{**}](\top \rightarrow \langle a^* \rangle \top)$. Thus, by Ax.r2, $\top \rightarrow \text{repeat}(a^*)$, and so $\text{repeat}(a^*)$ by MP. ■

In addition to Ax.r1 and Ax.r2 the next theorem requires Ax.?. Therefore, let L be any PDL that contains the logic P?r.

2.5.21 Theorem: $\vdash_L \text{repeat}(p?) \equiv p$.

Proof: Clearly $\text{repeat}(p?) \rightarrow \langle p? \rangle \text{repeat}(p?)$ by Ax.r1, therefore, by Ax.?, $\text{repeat}(p?) \rightarrow p \& \text{repeat}(p?)$. Thus, by PC, $\text{repeat}(p?) \rightarrow p$.

For the converse, first note that $p \rightarrow \langle p? \rangle p$ is equivalent to the tautology $p \rightarrow p$ by Ax.?. Therefore, $[(p?)^*](p \rightarrow \langle p? \rangle p)$ via RN, whence $p \rightarrow \text{repeat}(p?)$ by Ax.r2. ■

In addition to Ax.r1 and Ax.r2 the theorems 2.5.22 to 2.5.24 require Ax.c. Therefore, let L be any PDL that contains Pcr.

2.5.22 Theorem: $\vdash_L \text{repeat}((a+b)^c) \equiv \text{repeat}(a^c + b^c)$.

Proof: Let $p = \text{repeat}((a+b)^c)$. Then $p \rightarrow \langle (a+b)^c \rangle p$ (by Ax.r1), $p \rightarrow \langle a^c + b^c \rangle p$ (by 2.5.14a followed by Ax.+), and then $[(a^c + b^c)^*](p \rightarrow \langle a^c + b^c \rangle p)$ (by RN). Therefore, by Ax.r2,

$repeat((a+b)^c) \rightarrow repeat(a^c + b^c)$.

Likewise, if $p = repeat(a^c + b^c)$ then $[(a+b)^c](p \rightarrow \langle (a+b)^c \rangle p)$ (by Ax.r1, Ax.+, 2.5.14 and RN).

Thus, $repeat((a+b)^c) \rightarrow repeat(a^c + b^c)$ by Ax.r2. ■

2.5.23 Theorem: $\vdash_L repeat((ab)^c) \equiv repeat(b^c a^c)$.

Proof: Let $p = repeat((ab)^c)$. Then $p \rightarrow \langle (ab)^c \rangle p$ (by Ax.r1), $p \rightarrow \langle b^c a^c \rangle p$ (by 2.5.13a followed by Ax.;), and then $[(b^c a^c)^*](p \rightarrow \langle b^c a^c \rangle p)$ (by RN). Therefore, by Ax.r2, $repeat((ab)^c) \rightarrow repeat(b^c a^c)$.

Likewise, if $p = repeat(b^c a^c)$ then $[(ab)^c](p \rightarrow \langle (ab)^c \rangle p)$ (by Ax.r1, Ax.+, 2.5.13a and RN).

Thus, $repeat(b^c a^c) \rightarrow repeat((ab)^c)$ by Ax.r2. ■

2.5.24 Theorem: $\vdash_L repeat((a^*)^c)$.

Proof: According to 2.5.1b, $\top \rightarrow \langle a^* \rangle \top$. Thus, by 2.5.16a, $\top \rightarrow \langle a^{*c} \rangle \top$, and so by an application of RN we have $[a^{*c}](\top \rightarrow \langle a^{*c} \rangle \top)$. Consequently, by Ax.r2 and MP, $repeat(a^{*c})$. ■

The remaining theorems of this section require all of the axioms. Accordingly, let L be any PDL that contains P?cr.

2.5.25 Corollary: $\vdash_L repeat((p?)^c) \equiv p$.

Proof: (The proof is similar to that of 2.5.21.)

$repeat((p?)^c) \rightarrow p \& repeat((p?)^c)$ by Ax.r1, 2.5.17 and Ax.?. Thus, by PC, $repeat(p?) \rightarrow p$. Conversely, $[(p?)^c](p \rightarrow \langle (p?)^c \rangle p)$ by PT, Ax.? and 2.5.17, followed by RN. Thus $p \rightarrow repeat((p?)^c)$ by Ax.r2. ■

2.5.26 Theorem: $\vdash_L repeat((p?)^c) \equiv repeat(p?)$.

Proof: 2.5.21 and 2.5.25. ■

Notice, that within our list of immediate results, there is no theorem dealing with $repeat(ab)$ for arbitrary programs a and b . Actually, this situation coincides well with the formal semantics and interpretation of $repeat(a)$. It shall become clear in chapter three that there can be no valid statement about $repeat(ab)$ in general.

2.6 Maximality and Lindenbaum's Lemma

The results and notions of this section are vital for the model constructions of chapter four, where we set about to demonstrate completeness for the various systems of propositional dynamic logic introduced above.

Γ is a maximal extension of a propositional logic L if it is L -consistent and has only L -inconsistent proper extensions (i.e., if $p \notin \Gamma$ then $(p) \cup \Gamma$ is L -inconsistent). Let $\text{Max}_L \Gamma$ (or just $\text{Max } \Gamma$ when no ambiguities can arise) denote the fact that Γ is a maximal extension of L . According to Lindenbaum's lemma (which we prove below, 2.6.2) any consistent propositional logic may be extended to be maximally consistent. This fact, in particular, will figure prominently in the construction of canonical models - models that are used to invalidate non-theorems and (often) lead to completeness results for PDLs.

The following theorem states some basic facts about maximal extensions. (More often than not this theorem will be taken for granted in the proofs of chapter four.)

2.6.1 Theorem: Let L be a propositional logic and $\text{Max}_L \Gamma$. For $p, q \in \Sigma$

- (a) $p \in \Gamma$ iff $\vdash_L p$,
- (b) $L \subseteq \Gamma$,
- (c) $\top \in \Gamma$ and $\perp \notin \Gamma$,
- (d) $\neg p \in \Gamma$ iff $p \notin \Gamma$,
- (e) $p \vee q \in \Gamma$ iff $p \in \Gamma$ or $q \in \Gamma$,
- (f) $p \& q \in \Gamma$ iff $p \in \Gamma$ and $q \in \Gamma$,
- (g) $p \rightarrow q \in \Gamma$ iff (if $p \in \Gamma$ then $q \in \Gamma$),
- (h) $p \equiv q \in \Gamma$ iff ($p \in \Gamma$ if and only if $q \in \Gamma$),
- (i) if $p, p \rightarrow q \in \Gamma$ then $q \in \Gamma$ (i.e., Γ is closed under MP).

Proof:

(a) If $p \in \Gamma$ then $\vdash_L p$ is from 3.1(a). Conversely, assume that $\vdash_L p$ but that $p \notin \Gamma$ then $\text{Max}_L \Gamma$ implies that $\Gamma \cup \{p\}$ is L -inconsistent, i.e., $\vdash_L p \rightarrow \perp$, and so by PC, $\vdash_L \neg p$. By PC, it follows that $\vdash_L \neg p \& p$ or $\vdash_L \perp$ which contradicts the L -consistency of Γ .

- (b) Let $\vdash p$ then, in particular, $\Gamma \vdash p$, whence by (a), $p \in \Gamma$.
- (c) Since $\top \in L$ (by PC) then $\top \in \Gamma$ by (b); Also, if $\perp \in \Gamma$ then, by (a), Γ is L-inconsistent, which contradicts $\text{Max}_L \Gamma$.
- (d) $\Gamma \vdash p$ iff $\Gamma \vdash p \rightarrow \perp$ (by PC) iff $\Gamma \cup \{p\} \vdash \perp$ (by 2.3.1e) i.e. $\Gamma \cup \{p\}$ is L-inconsistent. Thus if $p \in \Gamma$ then Γ itself is L-inconsistent, which is false. On the other hand, since Γ has only L-inconsistent proper extensions, $p \notin \Gamma$ implies that $\Gamma \cup \{p\}$ is L-inconsistent.
- (e)-(h) Proof is straight forward from PC and earlier clauses.
- (i) Now if $\Gamma \vdash p \rightarrow q$ then $\Gamma \cup \{p\} \vdash q$ by 2.3.1e, so if $p \in \Gamma$ then we have $\Gamma \vdash q$ as required. ■

2.6.2 Lindenbaum's lemma: If Γ is L-consistent then there exists a set of formulae Δ such that $\Gamma \subseteq \Delta$ and $\text{Max}_L \Delta$.

Proof: We assume some exhaustive enumeration of Σ ,

$$p_1, p_2, p_3, \dots \text{ etc.}$$

and define a sequence of subsets of Σ containing Γ ,

$$\Delta_0, \Delta_1, \Delta_2, \Delta_3, \dots \text{ etc.}$$

where

$$\Delta_0 = \Gamma,$$

if $\Delta_n \cup \{p_{n+1}\}$ is L-consistent then $\Delta_{n+1} = \Delta_n \cup \{p_{n+1}\}$; else $\Delta_{n+1} = \Delta_n$.

By definition, each Δ_n is L-consistent. Let $\Delta = \bigcup_{n < \omega} \Delta_n$ so that Δ is automatically an extension of Γ (i.e., $\Gamma \subseteq \Delta$). We are required to show further that $\text{Max}_L \Delta$ i.e., (a) Δ is L-consistent and (b) that Δ has only L-inconsistent proper extensions.

(a) Suppose that Δ is L-inconsistent, then by 2.3.2, either L is inconsistent (which is contrary to Γ being L-consistent and so can be ignored) or there is a finite number of formulae, say $q_0, q_1, \dots, q_m \in \Delta$ such that $\vdash_L q_0 \& q_1 \& \dots \& q_m \rightarrow \perp$. Since each $q_k \in \Delta$, $0 \leq k \leq m$, and because $\Delta_j \subseteq \Delta_i$ for $0 \leq j \leq i$ then there is some l such that $q_k \in \Delta_l$, $0 \leq k \leq m$. Therefore Δ_l is itself L-inconsistent which is false. Thus Δ must be L-consistent.

(b) Now suppose that $\Delta \cup \{p\}$ is L-consistent for some formula $p \in \Sigma$. For some j , p appears as the j^{th} formula of our enumeration, i.e., $p = p_j$. Since $\Delta \cup \{p_j\}$ is L-consistent then so is $\Delta_{j-1} \cup \{p_j\}$ (by 2.3.3). In this case, $\Delta_{j-1} \cup \{p_j\} = \Delta_j$. Since $\Delta_j \subseteq \Delta$ then $p = p_j \in \Delta$, and so Δ has no L-consistent proper extensions. ■

2.6.3 Corollary: If L is consistent then there exists a set Δ such that $\text{Max}_L \Delta$.

Proof: Set $\Gamma = \emptyset$ in Lindenbaum's lemma. ■

2.6.4 Theorem: Let N be a normal propositional logic and let $\text{Max}_N \Gamma$ and $\text{Max}_N \Upsilon$, then,
for any $a \in \Pi$, $(p \mid [a]p \in \Gamma) \subseteq \Upsilon$ iff $\langle a \rangle p \mid p \in \Upsilon \subseteq \Gamma$.

Proof: Suppose that $(p \mid [a]p \in \Gamma) \subseteq \Upsilon$. Let $p \in \Upsilon$ then $\neg p \notin \Upsilon$ and so $[a]\neg p \in \Gamma$ or $\neg \langle a \rangle p \in \Gamma$.
Therefore $\langle a \rangle p \in \Gamma$. Conversely, suppose that $\langle a \rangle p \mid p \in \Upsilon \subseteq \Gamma$. Let $[a]p \in \Gamma$ then $\langle a \rangle \neg p \in \Gamma$.
Thus, $\neg p \notin \Upsilon$ or $p \in \Upsilon$. ■

Chapter 2 Notes:

acronym

In the literature of dynamic logic the acronym PDL is usually reserved for the sentential version of dynamic logic where programs are considered to be regular expressions supplemented with tests (which we also define below). However, we let PDL refer to any propositional dynamic logic, including test free versions, where programs are taken to be regular expressions over Π_0 .

1st-order decidability

The presence of iteration prevents there being any effective axiomatisation of first-order dynamic logics or QDL's, and for this reason they are undecidable (see [HAR79;pp.23-25] and, for a more detailed account, [M&D82]). However, since all the PDLs we consider are finitely axiomatisable (and because most of them satisfy the finite model property) then they are also decidable (chapter three, section 3.5).

induction axiom

IN is "a kind of induction schema . . . the counterparts in tense logic [are] due to Dana Scott and E.J. Lemmon" [SEG78;pp.36] and are discussed by Arthur Prior in his book *Past, Present and Future*, Clarendon Press, Oxford 1967. See pp.66 ff).

deontic logic

A basic deontic principle is that "what ought to be done is permissible". Using the syntax of modal logic where $\Box$ and $\Diamond$ are used to designate the deontic concepts "obligation" and "permission" this may be formalized as the axiom schema

$$(D) \vdash \Box p \rightarrow \Diamond p.$$

For any normal modal propositional logic, D is equivalent to the rule

$$\vdash p \text{ implies } \vdash \Diamond p,$$

which is the counterpart to our rule RP in modal logic. Consult [CHE80;pp.133] for proof.

<a> deontic modality*

Thus, we have the rule: if $\vdash p$ then $\vdash \langle a^* \rangle p$, " a^* -possiblization" and so $\langle a^* \rangle$ is a deontic modality. See the above note *deontic logic*. Note that, by combining 2.5.1a and b (by PC) we also have a subspecies of the basic deontic axiom D of (NOTE: deontic logic).

[a] S4 modality*

Thus, in the well established terminology of modal logic, 2.5.1 and 2.5.4 make it clear that $[a^*]$ is an S4-modality.

CHAPTER THREE

SEMANTICS

In this chapter the basic set theoretic semantics for nondeterministic propositional dynamic logic is formalized and the intuitive countable model property verified. We demonstrate the soundness of our various versions of propositional dynamic logic with respect to the standard semantics and then show that even the most elementary version of PDL is not semantically compact (even when a stronger than usual notion of semantical consequence is used to define compactness). Finally, we discuss properties of PDL to have models of "manageable" size: the collapsing, finite model and tree model properties. The importance of models of a "manageable" size will be seen in the last section where we briefly consider the decidability of all the PDL under our consideration.

3.1 Standard and Non-Standard Kripke Models

In the most general semantics for propositional dynamic logics the interpretations of all programs, whether they are basic or compound, are left totally unconstrained. The logical operators, i.e., **OpForm(modal)**, are given their usual meaning. A (π -unrestricted) model is a triple (W, π, ν) where (i) W is a non-empty set called the set of (execution) states, (ii) π is a function mapping each well-formed program into a binary relation on W , i.e., $\pi : \Pi \longrightarrow \mathcal{P}(W^2)$, (iii) ν is a function mapping each $P \in \Sigma_0$ into a subset of W , i.e., $\nu : \Sigma_0 \longrightarrow \mathcal{P}(W)$.

Let $(u, v) \in \pi(a)$. We say that a program a starting in state u (called the input, initial

or starting state), can execute, terminating in state v (called the output, final or terminating state). The ordered pair (u,v) is called a transition of program a from starting state u to terminating state v . Note that, the above interpretation of a program as a binary relation prevents any "direct" way of saying that it can make infinite progress without terminating. "The semantics appears to leave no trace of diverging computations." [PRA80b;pp.268] Either $(u,v) \in \pi(a)$ or $(u,v) \notin \pi(a)$, and that is about it.

For all models ν is automatically extended inductively to satisfy the following conditions:

$$\begin{aligned}\nu(\perp) &= \emptyset, \\ \nu(\neg p) &= W - \nu(p), \\ \nu(p \vee q) &= \nu(p) \cup \nu(q), \\ \nu(\langle a \rangle p) &= \{w \in W \mid \exists v ((w,v) \in \pi(a) \text{ and } v \in \nu(p))\}.\end{aligned}$$

and, if $\Sigma_{\text{PDL,repeat}} \subseteq \Sigma$ (i.e., $\text{OpForm} = \text{OpForm}(\text{modal,repeat})$) then, for all $a \in \Pi$,

$$\begin{aligned}(\text{ill-founded}) \quad \nu(\text{repeat}(a)) &= \{w_0 \in W \mid \exists w_1, \exists w_2, \dots \text{ such that, for} \\ &\quad \text{all } i \geq 0, (w_i, w_{i+1}) \in \pi(a)\}.\end{aligned}$$

(NOTE: well-defined relations) Thus, $\text{repeat}(a)$ is mapped by ν into the set of starting states that allow program a to execute repeatedly, infinitely often. (NOTE: semantics of *loop*)

Let $\circ$ be relative product (i.e., relational composition, for relations S and T over field W , $S \circ T = \{(x,y) \mid \exists z \text{ such that } (x,z) \in S \text{ and } (z,y) \in T\}$) and, for a relation S , we define S^n inductively as

$$\begin{aligned}S^0 &= \{(x,x) \mid x \in W\}, \\ S^{n+1} &= S \circ S^n.\end{aligned}$$

A (π -unrestricted) model, say $\mathfrak{S} = (W, \pi, \nu)$, is called a Kripke model if π satisfies the

following constraints for any $a, b \in \Pi$:

- (union) $\pi(a+b) = \pi(a) \cup \pi(b)$,
- (composition) $\pi(ab) = \pi(a) \circ \pi(b)$,
- (*-reflexive) $\forall u \in W, (u, u) \in \pi(a^*)$ i.e., $\pi(a)^0 \subseteq \pi(a^*)$,
- (*-inherit) $\pi(a) \subseteq \pi(a^*)$,
- (*-transitive) $\pi(a^*) \circ \pi(a^*) \subseteq \pi(a^*)$,
- (λ -identity) $\pi(\lambda) = \{(w, w) \mid w \in W\}$ i.e., $\forall a \in \Pi_{PDL}, \pi(\lambda) = \pi(a)^0$,

and if $\Pi_{PDL, ?} \subseteq \Pi$

- (test) $\pi(p?) = \{(w, w) \mid w \in \nu(p)\}$

and if $\Pi_{PDL, c} \subseteq \Pi$

- (inverse) $\pi(a^c) = \pi(a)^{-1} = \{(u, v) \mid (v, u) \in \pi(a)\}$

If in addition to the above constraints π also satisfies:

- (*-ancestral) $\pi(a^*) = \bigcup_{n < \omega} \pi(a)^n$

then $\mathfrak{S}$ is called a standard Kripke model. Otherwise $\mathfrak{S}$ is called a non-standard Kripke model. It is obvious from the above definitions that if a model satisfies the *-ancestral condition then it also satisfies the *-reflexive, *-inherit and *-transitive conditions. Also, $\mathfrak{S}$ is non-standard just in case it satisfies:

- (*-non-standard) $\pi(a^*) \supset \bigcup_{n < \omega} \pi(a)^n$

(i.e., the ancestral or reflexive transitive closure of $\pi(a)$ is a proper subset of $\pi(a^*)$). If a model satisfies the *-non-standard condition then it also satisfies the *-reflexive, *-inherit and *-transitive conditions.

Recall that in chapter two, non-tautologous theorems specifying an equivalent to $repeat(ab)$ for arbitrary program a and b alluded us. We can now see this situation reflected in the semantics for $repeat(ab)$. For instance, it is simple enough to give two programs, say a and b , that when combined by relational composition, $\pi(a) \circ \pi(b)$, satisfy $repeat(ab)$ but for which $repeat(a)$ and $repeat(b)$ are not true (e.g., let $\pi(a) = \{(u, v)\}$ and $\pi(b) = \{(v, u)\}$). Conversely, it is possible that $repeat(a)$ and $repeat(b)$ but not $repeat(ab)$ (e.g., let $\pi(a) = \{(u, v), (v, u)\}$ and $\pi(b) = \{(v, v)\}$). In fact, there seems to be no specifiable conditions on arbitrary a and b (short of $Ax.r2$ or its corollaries) that will allow us infer $repeat(ab)$.

Notice that, given any W , $\pi_0: \Pi_0 \longrightarrow \mathcal{P}(W^2)$ and $\nu: \Sigma_0 \longrightarrow \mathcal{P}(W)$, that we can extend π_0 inductively to a mapping, say π_* (i.e., π_* covers all programs, both basic and compound), so that $\mathfrak{S} = (W, \pi_*, \nu)$ is a standard Kripke model. (The required conditions of standard Kripke models are merely stipulated.) Thus, it is sufficient, when constructing a standard Kripke model, to provide W , π_* restricted to Π_0 and ν restricted to Σ_0 .

In classical logic the notion of logical truth is based on the straightforward concept of "truth-in-a-model". Since Kripke [KRI63] developed possible world semantics, however, it has been necessary, at least in the context of modal and dynamic logics, to recognize an underlying refinement of "truth-in-a-model". The most basic level of logical truth is the "at-a-state" version. Let $\mathfrak{S} = (W, \pi, \nu)$ be a model. The basic level of logical truth is the "at-a-state" version. We write $\mathfrak{S} \models_w p$ if $w \in \nu(p)$ and say that p is locally true at state w in $\mathfrak{S}$ or p is (locally) satisfied at state w in $\mathfrak{S}$ (and in this case we sometimes say that $\mathfrak{S}$ is a model of p). The second level of logical truth is the usual "in-a-model" version. We say that p is globally true in $\mathfrak{S}$, globally satisfied in $\mathfrak{S}$ or just true in $\mathfrak{S}$ if, for all $w \in W$, $\mathfrak{S} \models_w p$. Thus, p is (globally) true in $\mathfrak{S}$ whenever p is satisfied everywhere in $\mathfrak{S}$. Satisfaction (of the local variety) is the semantical counterpart of consistency. (For example, if a Σ_{PDL} formula p is true at some state in some model then the p is P-consistent.) Where p is (globally) true in $\mathfrak{S}$ we simply write $\mathfrak{S} \models p$. If Γ is a set of formulae, we write $\mathfrak{S} \models \Gamma$ if and only if $\mathfrak{S} \models p$ for all $p \in \Gamma$. Similarly, we write $\mathfrak{S} \models \Gamma$ if and only if $\mathfrak{S} \models p$ for all $p \in \Gamma$.

Let Ω be a class of models and suppose that $\mathfrak{S} = (W, \pi, \nu) \in \Omega$. If $\mathfrak{S} \models_w p$ for some $w \in W$ then formula p is said to be locally Ω -satisfied (by $\mathfrak{S}$ at w) or just Ω -satisfied (by $\mathfrak{S}$ at w). If $\mathfrak{S} \models p$ then formula p is said to be globally Ω -satisfied (by $\mathfrak{S}$). Note that global Ω -satisfiability implies local Ω -satisfiability but not the converse. For instance, the Σ_{PDL} formula $p \& \neg [a]p$ is P -consistent (and therefore, by the completeness of P with respect to SKM (say), it is SKM-satisfiable) but it is not generally globally true for a SKM model. Finally, we define what it means for a set of formulae to be Ω -satisfied. If Γ is a set of formulae and $\mathfrak{S} \models_w \Gamma$ for some $w \in W$ then Γ is said to be (locally) Ω -satisfied (by $\mathfrak{S}$). Likewise, if $\mathfrak{S} \models \Gamma$ then Γ is said to be globally Ω -satisfied (by $\mathfrak{S}$).

In addition to the two aforementioned notions of "truth-in-a-model", the top level of truth is that of validity. Let Ω be a class of models. If p is true for all $\mathfrak{S} \in \Omega$ we say that p is Ω -valid i.e., p is Ω -valid if and only if p is locally true at all states in all Ω type models.

The layered approach to logical truth made necessary by our adoption of possible world semantics will carry over into other semantical notions. Thus, for example, based on the two distinct notions of truth defined above, we must also differentiate between local and global versions of logical consequence and semantical compactness. We cover these briefly in section 3.3.

3.1.1 Theorem: For any model $\mathfrak{S} = (W, \pi, \nu)$ and any $w \in W$

- a) not $\mathfrak{S} \models_w \perp$,
- b) $\mathfrak{S} \models_w \neg p$ iff not $\mathfrak{S} \models_w p$,
- c) $\mathfrak{S} \models_w p \vee q$ iff $\mathfrak{S} \models_w p$ or $\mathfrak{S} \models_w q$,
- d) $\mathfrak{S} \models_w \langle a \rangle p$ iff $\exists v \in W$ such that $(w, v) \in \pi(a)$ and $\mathfrak{S} \models_v p$,
- e) $\mathfrak{S} \models_w [a]p$ iff $\forall v \in W$, $(w, v) \in \pi(a)$ implies that $\mathfrak{S} \models_v p$,
- f) $\mathfrak{S} \models_w p \& q$ iff $\mathfrak{S} \models_w p$ and $\mathfrak{S} \models_w q$,
- g) $\mathfrak{S} \models_w p \rightarrow q$ iff (if $\mathfrak{S} \models_w p$ then $\mathfrak{S} \models_w q$),
- h) $\mathfrak{S} \models_w p \equiv q$ iff ($\mathfrak{S} \models_w p$ iff $\mathfrak{S} \models_w q$).

Proof:

For a) note that $\nu(\perp) = \emptyset$, so we can never have $\mathfrak{S} \models_w \perp$.

For b) $\mathfrak{S} \models_w \neg p$ iff $w \in \nu(\neg p)$ iff $w \in W - \nu(p)$ iff $w \notin \nu(p)$ iff not $\mathfrak{S} \models_w p$.

For c) $\mathfrak{S} \models_w p \vee q$ iff $w \in \nu(p \vee q)$ iff $w \in \nu(p) \cup \nu(q)$ iff $w \in \nu(p)$ or $w \in \nu(q)$ iff $\mathfrak{S} \models_w p$ or $\mathfrak{S} \models_w q$.

For d) $\mathfrak{S} \models_w \langle a \rangle p$ iff $w \in \nu(\langle a \rangle p)$ iff $\exists v \in W$ such that $(w, v) \in \pi(a)$ and $v \in \nu(p)$ iff $\exists v \in W$ such that $(w, v) \in \pi(a)$ and $\mathfrak{S} \models_v p$.

For e) $\mathfrak{S} \models_w [a] p$ iff not $\mathfrak{S} \models_w \langle a \rangle \neg p$ iff not $\exists v \in W$ such that $(w, v) \in \pi(a)$ and not $\mathfrak{S} \models_v p$ (by d and b) iff $\forall v \in W, (w, v) \in \pi(a)$ implies $\mathfrak{S} \models_v p$.

For f), g) and h) expand and use the preceding clauses. ■

Let KM be the class of Kripke models and let SKM and non-SKM partition KM into the sets of standard and non-standard Kripke models respectively. Another useful, albeit syntactically motivated, class of models are the so-called induction or loop invariant models. (NOTE: induction model history) An *induction model* is a Kripke model where each instance of $Ax.ind$ is true. We denote this class of models by INM . What is interesting about induction models is that no formula of $\Sigma_{PDL, ?, \text{repeat}}$ can distinguish them from standard Kripke models (chapter four). However, despite the fact that all standard Kripke models are induction models (3.1.3), there are induction models which are not in SKM (3.1.2).

3.1.2 Theorem: There are non-standard induction models.

Proof: Let N be the set of natural numbers and let N^i be a finite subset of N (the superscript i gives the place of N^i in some assumed enumeration of the finite subsets of N). Consider the following induction model $\mathfrak{S} = (W, \pi, \nu)$ where

$$W = W_x \cup W_y; \quad W_x = \{x_i\}_{i>0}; \quad W_y = \{y_i\}_{i>0},$$

for all $P_j \in \Sigma_0$,

$$\nu(P_j) = \nu_x \cup \nu_y; \quad \nu_x = \{x_i \mid j \in N^i\}; \quad \nu_y = \{y_i \mid j \in N^i\},$$

for distinguished basic program A ,

$$\pi(A) = \pi_x(A) \cup \pi_y(A); \quad \pi_x(A) = \{(x_i, x_j) \mid \text{for all } i, j\}; \quad \pi_y(A) = \{(y_i, y_j) \mid \text{for all } i, j\},$$

$$\pi(A^*) = W^2; \quad \pi_x(A^*) = W_x^2; \quad \pi_y(A^*) = W_y^2,$$

and for any program b (not A)

$$\text{if } b \in \Pi_0 \text{ then: } \pi(b) = \emptyset; \quad \pi_x(b) = \emptyset; \quad \pi_y(b) = \emptyset,$$

$$\text{if } b \notin \Pi_0 \text{ then: } \pi(b^*) = \bigcup_{i < \omega} \pi(b)^i; \quad \pi_x(b^*) = \bigcup_{i < \omega} \pi_x(b)^i; \quad \pi_y(b^*) = \bigcup_{i < \omega} \pi_y(b)^i.$$

In the case of compound programs other than A^* we assume the standard extension. Therefore $\mathfrak{S}$ acts as if it is a standard Kripke model except in the case of program A^* . Observe that $\mathfrak{S}$ is non-standard at A^* since there are no $n \in \mathbb{N}$, x_i and y_j such that $(x_i, y_j) \in \pi(A)^n$.

The reader can regard $\mathfrak{S}$ as two separate and well-behaved infinite standard Kripke models, $\mathfrak{S}_x = (W_x, \pi_x, \nu_x)$ and $\mathfrak{S}_y = (W_y, \pi_y, \nu_y)$, where all states of one model are connected with all of the other model by the program A^* .

Now we show that $\mathfrak{S}$ is an induction model: Since $\mathfrak{S}$ is standard except in regards to the compound program A^* , we need only show that, for any $p \in \Sigma$, $\nu(p^{\text{ind}}) = W$, where p^{ind} denotes the formula $\langle A^* \rangle p \rightarrow p \vee \langle A^* \rangle (\neg p \ \& \ \langle A \rangle p)$. Notice that $x_i \in \nu_x(p)$ iff $y_i \in \nu_y(p)$ - which means that $\mathfrak{S}_x$ and $\mathfrak{S}_y$ cannot be distinguished by any formula of $\Sigma_{\text{PDL}, ?, c, \text{repeat}}$. In particular, p^{ind} is true in both $\mathfrak{S}_x$ and $\mathfrak{S}_y$, and so it must be true in $\mathfrak{S}$ also. ■ (NOTE: 3.1.2 proof)

3.1.3 Theorem: $\text{SKM} \subseteq \text{INM}$.

Proof: Let p_{ind} be an instance of $Ax.\text{ind}$. First we show that, if $\mathfrak{S} = (W, \pi, \nu) \in \text{SKM}$ then $\nu(p_{\text{ind}}) = W$. Suppose that $w \in \nu(\langle a^* \rangle p)$. Through 3.1.1, we are required to show that $w \in \nu(p)$ or $w \in \nu(\langle a^* \rangle (\neg p \ \& \ \langle a \rangle p))$. By 3.3.1d and since π satisfies the ancestral condition, there is an m such that $(w, x) \in \pi(a)^m$ and $x \in \nu(p)$. Let n be the minimum such m . If $n=0$ then we are done, since in this case $x=w$. If we assume that $n \geq 1$ (i.e., not $x=w$) then, there is a $y \in W$ such that $(w, y) \in \pi(a)^{n-1}$, $(y, x) \in \pi(a)$ and, because n is minimum, $y \notin \nu(p)$. And, since $(w, y) \in \pi(a)^{n-1}$, also $(w, y) \in \pi(a^*)$ (by the "standardness" of π). Thus $w \in \nu(\langle a^* \rangle (\neg p \ \& \ \langle a \rangle p))$. Consequently, $\text{SKM} \subseteq \text{INM}$.

Finally, since there are induction models which are non-standard (3.1.2) it must be that INM properly contains SKM . ■

We say that $\mathfrak{S}' = (W', \pi', \nu')$ is a Ω -substructure of a model $\mathfrak{S} = (W, \pi, \nu) \in \Omega$ if $W' \subseteq W$, $\mathfrak{S}' \models_{\omega} p$ implies $\mathfrak{S} \models_{\omega} p$ and $\mathfrak{S}' \in \Omega$. Observe that if a formula is satisfiable in a model then it can be satisfied by a countable substructure of that model. Intuitively, this is because our programs can visit at most countably many states in the process of their execution. Indeed, if it wasn't for our consideration of *repeat*, programs could visit only a finite number of execution states before halting. For the remainder of this section, let Ω be

one of KM, SKM, non-SKM or INM.

3.1.4 Theorem: If $\mathfrak{S}=(W,\pi,\nu)\in\Omega$ and $\mathfrak{S}\models_{\mathbf{w}}p$ then there is a countable Ω -substructure of $\mathfrak{S}$, say $\mathfrak{S}'$, such that $\mathfrak{S}'\models_{\mathbf{w}}p$.

Proof: By structural complexity of p .

Case 1 - $p=P\in\Sigma_0$: Clearly, if $\mathfrak{S}'=((w),\pi',\nu')$, where π' and ν' are suitably restricted (i.e., $\nu':\Sigma_0\rightarrow\mathcal{P}(\{w\})$ and $\pi':\Pi\rightarrow\emptyset$), then $\mathfrak{S}'\models_{\mathbf{w}}P$.

Case 2 - $p=\neg q$: Obviously, if not $\mathfrak{S}\models_{\mathbf{w}}q$ (3.1.1b) then, for any substructure $\mathfrak{S}'$ of $\mathfrak{S}$, not $\mathfrak{S}'\models_{\mathbf{w}}q$ also. Therefore (by 3.1.1b) $\mathfrak{S}'\models_{\mathbf{w}}\neg q$.

Case 3 - $p=q\vee r$: By the induction hypothesis there are countable substructures of $\mathfrak{S}$, say $\mathfrak{S}_1=(W_1,\pi_1,\nu_1)$ and $\mathfrak{S}_2=(W_2,\pi_2,\nu_2)$, such that $\mathfrak{S}_1\models_{\mathbf{w}}q$ and $\mathfrak{S}_2\models_{\mathbf{w}}r$. Let $\mathfrak{S}'=(W_1\cup W_2,\pi_1\cup\pi_2,\nu_1\cup\nu_2)$. Then, it is clear that $\mathfrak{S}'$ is a countable substructure of $\mathfrak{S}$ such that $\mathfrak{S}'\models_{\mathbf{w}}q$ and $\mathfrak{S}'\models_{\mathbf{w}}r$, i.e., $\mathfrak{S}'\models_{\mathbf{w}}q\vee r$ (by 3.1.1c).

Case 4 - $p=\langle a \rangle q$: Suppose that $(w,v)\in\pi(a)$ and $\mathfrak{S}\models_v q$ (as by 3.3.1d). By the induction hypothesis, there is a countable substructure $\mathfrak{S}''\models_v q$ where, say, $\mathfrak{S}''=(W'',\pi'',\nu'')$. Let $\mathfrak{S}'=(W''\cup\{w\},\pi',\nu')$ where π' is just as π'' but extended (if required) so that $(w,v)\in\pi'(a)$ - obviously this is possible since $(w,v)\in\pi(a)$. Then $\mathfrak{S}'\models_{\mathbf{w}}\langle a \rangle q$ (3.3.1d).

Case 5 - $p=\text{repeat}(a)$: Suppose that $w\in\nu(\text{repeat}(a))$, then, by definition there are $w_0=w, w_1, w_2, \dots$, etc., such that $(w_i, w_{i+1})\in\pi(a)$ for $0\leq i<\omega$. Let $W'=\{w_i\}_{0\leq i<\omega}$, $\pi'(a)=\{(w_i, w_{i+1})\}_{0\leq i<\omega}$ and undefined otherwise, and let ν' be ν suitably restricted (i.e., $\nu':\Sigma_0\rightarrow W'$ and undefined otherwise).

In each of the above cases, it is obvious that $\mathfrak{S}'\in\Omega$. ■

A quick perusal of the proof of the above theorem shows that if *repeat* is absent from p then there are finite Ω -substructures which satisfy p . Thus, it is only the semantics of the *repeat* operator that (possibly) requires infinite models. Actually, in section 3.4 we give a formula which combines *repeat* with the converse operator so that it can have no finite model.

3.1.5 Corollary: Let $p\in\Sigma_{\text{PDL},\tau,c}$. If $\mathfrak{S}=(W,\pi,\nu)\in\Omega$ and $\mathfrak{S}\models_{\mathbf{w}}p$ then there is a finite Ω -substructure of $\mathfrak{S}$, say $\mathfrak{S}'$, such that $\mathfrak{S}'\models_{\mathbf{w}}p$.

Also in section 3.4 we prove a much stronger result: the so-called collapsed Ω -model property for PDL P , $P?$, Pc and $P?c$. The strength of this result is in the fact that it shows us how to build a finite model that satisfies p when $\mathfrak{S}$ satisfies p . (The resulting model, however, is not a substructure of $\mathfrak{S}$). The collapsed Ω -model result is further strengthened in chapter four where finite models are constructed from canonical models.

3.2 Soundness and Consistency

Let L be one of the logics P , $P?$, Pc , Pr , $P?c$, $P?r$, Pcr or $P?cr$. One of the major objectives of this thesis is to prove that p is a theorem of the logic L if and only if p is SKM-valid or INM-valid. That is, the logic L coincides with the class of formulae true for all standard Kripke or all induction models. These facts combine to show that no expression of the PDL languages defined in chapter two is able to distinguish between at least two different classes of models. Completeness, i.e., that if p is SKM or INM-valid then p is a theorem of L , is difficult to prove and will be taken up in chapter four. Before doing so, we derive the more straightforward soundness results, after which consistency results fall out as corollaries.

We show that each PDL considered in this thesis is sound with respect to INM and SKM, but fails to be sound with respect to non-SKM and KM. The demonstration of soundness is lemmatical: First we show that any instance of each of our axiom schema is valid and each of our rules of inference preserves validity (theorems 3.2.1 through to 3.2.10). These results combine to provide the inductive proof of soundness (theorem 3.2.11). This initial step of the inductive proof is based on the fact that the axioms are valid. The inductive step is based on the fact that all of our rules of inference take valid formulae to other valid formulae.

3.2.1 Theorem: For any model $\mathfrak{S}=(W, \pi, \nu)$, if $\mathfrak{S} \models p$ and $\mathfrak{S} \models p \rightarrow q$ then $\mathfrak{S} \models q$ (i.e., modus ponens preserves validity in $\mathfrak{S}$).

Proof: This is a direct consequence of 3.1.1g. ■

3.2.2 Theorem: Instances of Ax_K are true in any model.

Proof: Let $\mathcal{S}=(W,\pi,\nu)$ be a Kripke model and let $w \in W$. If $\mathcal{S} \models_w [a](p \rightarrow q)$, $\mathcal{S} \models_w [a]p$ and $(w,v) \in \pi(a)$ then $\mathcal{S} \models_v (p \rightarrow q)$ and $\mathcal{S} \models_v p$ (by 3.1.1e). So, by 3.1.1g, $\mathcal{S} \models_v q$. Hence $\mathcal{S} \models_w [a]q$ (by 3.1.1e) and then $\mathcal{S} \models_w [a]p \rightarrow [a]q$ (by 3.1.1g). Finally (by 3.1.1g once again), $\mathcal{S} \models_w [a](p \rightarrow q) \rightarrow ([a]p \rightarrow [a]q)$. ■

3.2.3 Theorem: For any model $\mathcal{S}=(W,\pi,\nu)$ and $a \in \Pi_{PDL}$, if $\mathcal{S} \models p$ then $\mathcal{S} \models [a]p$ (i.e., a-necessitation preserves validity in $\mathcal{S}$).

Proof: Let $w \in W$ and suppose that $(w,u) \in \pi(a)$. Now $\mathcal{S} \models_u p$ (by definition) and therefore $\mathcal{S} \models_w [a]p$ (by 3.1.1e). ■

3.2.4 Theorem: Instances of Ax_+ are true in any model that satisfies the union condition.

Proof: Let $\mathcal{S}=(W,\pi,\nu)$ be any model that satisfies the union condition and let $w \in W$. By 3.1.1h we are required to show that $\mathcal{S} \models_w \langle a+b \rangle p$ iff $\mathcal{S} \models_w \langle a \rangle p \vee \langle b \rangle p$.

$\mathcal{S} \models_w \langle a+b \rangle p$ iff $\exists v \in W$ such that $(w,v) \in \pi(a+b)$ and $\mathcal{S} \models_v p$ (3.1.1d)
iff $\exists v \in W$ such that $((w,v) \in \pi(a) \text{ or } (w,v) \in \pi(b))$ and $\mathcal{S} \models_v p$ (union condition)
iff $\mathcal{S} \models_w \langle a \rangle p \vee \langle b \rangle p$ (3.1.1d; 3.1.1c). ■

3.2.5 Theorem: Instances of $Ax_;$ are true in any model that satisfies the composition condition.

Proof: Let $\mathcal{S}=(W,\pi,\nu)$ be any model that satisfies the composition condition and let $w \in W$. By 3.1.1h we are required to show that $\mathcal{S} \models_w \langle ab \rangle p$ iff $\mathcal{S} \models_w \langle a \rangle \langle b \rangle p$.

$\mathcal{S} \models_w \langle ab \rangle p$ iff $\exists v \in W$ such that $(w,v) \in \pi(ab)$ and $\mathcal{S} \models_v p$ (3.1.1d)
iff $\exists z, v \in W$ such that $(w,z) \in \pi(a)$ and $(z,v) \in \pi(b)$ and $\mathcal{S} \models_v p$ (composition condition)
iff $\exists z \in W$ such that $(w,z) \in \pi(a)$ and $\mathcal{S} \models_z \langle b \rangle p$ (3.1.1d)
iff $\mathcal{S} \models_w \langle a \rangle \langle b \rangle p$ (3.1.1d). ■

3.2.6 Theorem: Instances of Ax_* are true in any model that satisfies the $*$ -inherit, $*$ -

reflexive and $*$ -transitive conditions.

Proof: Let $\mathcal{S}=(W,\pi,\nu)$ be a model that satisfies the $*$ -inherit, $*$ -reflexive and $*$ -transitive conditions and let $w \in W$. By 3.1.1h and 3.1.1c we are required to show that if $\mathcal{S} \models_w p$ or $\mathcal{S} \models_w \langle a \rangle \langle a^* \rangle p$ then $\mathcal{S} \models_w \langle a^* \rangle p$.

Assume $\mathcal{S} \models_w p$. Since π is $*$ -reflexive then $(w,w) \in \pi(a^*)$. Hence, by 3.1.1d, $\mathcal{S} \models_w \langle a^* \rangle p$. On the other hand let $\mathcal{S} \models_w \langle a \rangle \langle a^* \rangle p$, i.e., by 3.1.1d, $\exists v, z \in W$ such that $(w,v) \in \pi(a)$, $(v,z) \in \pi(a^*)$ and $\mathcal{S} \models_v p$. Since π satisfies the $*$ -inherit condition then $(w,v) \in \pi(a^*)$, and therefore by the $*$ -transitive quality of π we have $(w,z) \in \pi(a^*)$. Consequently, since also $\mathcal{S} \models_v p$ then $\mathcal{S} \models_w \langle a^* \rangle p$. ■

3.2.7 Theorem: Instances of Ax.id are true in any model that satisfies the λ -identity condition.

Proof: Let $\mathcal{S}=(W,\pi,\nu)$ be a Kripke model. Since π satisfies the λ -identity condition then for all $w \in W$, $\mathcal{S} \models_w \langle \lambda \rangle p$ iff $\mathcal{S} \models_w p$ (3.1.1d). Thus by repeated use of 3.1.1h, $\mathcal{S} \models_w \langle \lambda \rangle p \equiv p$. ■

3.2.8 Theorem: Assume that $\Pi_{PDL,?} \subseteq \Pi$. Instances of Ax.?¹ are true in any model that satisfies the test condition.

Proof: Let $\mathcal{S}=(W,\pi,\nu)$ be a Kripke model. $\mathcal{S} \models \langle p? \rangle q$ iff $\forall w \in W, \exists v \in W$ such that $(w,v) \in \pi(p?)$ and $w \in \nu(q)$ (by 3.1.1d) iff, since π satisfies the test condition, $\forall w \in W, w \in \nu(p)$ and $w \in \nu(q)$ iff $\mathcal{S} \models p \& q$ (by 3.1.1f). ■

3.2.9 Theorem: Assume that $\Pi_{PDL,c} \subseteq \Pi$. Instances of Ax.c are true in any model that satisfies the inverse condition.

Proof: Let $\mathcal{S}=(W,\pi,\nu)$ be a Kripke model. Let $\mathcal{S} \models_w p$ and suppose that π satisfies the inverse condition. Notice that if $(w,v) \in \pi(a)$ then $(v,w) \in \pi(a^c)$, and, if $(w,v) \in \pi(a^c)$ then $(v,w) \in \pi(a)$. By 3.1.1d, followed by 3.1.1e, applied to both of these facts we have that (a) $\mathcal{S} \models_w [a] \langle a^c \rangle p$ and (b) $\mathcal{S} \models_w [a^c] \langle a \rangle p$. The final result follows by 3.1.1. ■

3.2.10 Theorem: Assume that $\Pi_{PDL,repeat} \subseteq \Pi$.

- (a) Instances of Ax.r1 are true in any model that satisfies the ill-founded condition.

(b) Instances of $Ax.r2$ are true in any model that satisfies the ill-founded and $*$ -reflexive conditions.

Proof: Let $\mathfrak{S}=(W, \pi, \nu)$ be a Kripke model.

(a) Let $\mathfrak{S} \models_w repeat(a)$, i.e., $w \in \nu(repeat(a))$. This is tantamount to $\exists w_0, w_1, \dots$ etc. such that $w_0 = w$ and, for all $i \geq 0$, $(w_i, w_{i+1}) \in \pi(a)$, which is the same as $(w, w_1) \in \pi(a)$ and $\exists w_1, w_2, \dots$, etc. such that, for all $i \geq 1$, $(w_i, w_{i+1}) \in \pi(a)$. This last condition means that $\mathfrak{S} \models_w \langle a \rangle repeat(a)$, as required.

(b) Suppose that $\mathfrak{S} \models_w [a^*](p \rightarrow \langle a \rangle p)$. This means, by 3.1.1, that for all $v \in W$, $(w, v) \in \pi(a^*)$ implies that if $v \in \nu(p)$ then there is a $u \in W$ such that $(v, u) \in \pi(a)$ and $u \in \nu(p)$. Now suppose that $w \in \nu(p)$. Because of the $*$ -reflexive condition of $\pi(a^*)$ we have that $(w, w) \in \pi(a^*)$ and therefore there is a $w_1 \in W$, such that $(w, w_1) \in \pi(a)$ and $w_1 \in \nu(p)$. Similarly, we have $w_2, w_3, \dots$, etc. such that, for all $i \geq 1$, $(w_i, w_{i+1}) \in \pi(a)$. Setting $w = w_0$, we have $w \in \nu(repeat(a))$ and, hence, by 3.1.1g, $\mathfrak{S} \models_w p \rightarrow repeat(a)$. A further application of 3.1.1g gives the required result. ■

3.2.11 The Soundness theorem for logic $P, P?, Pc, Pr, P?c, P?r, Pcr$, and $P?cr$: If L is one of $P, P?, Pc, Pr, P?c, P?r, Pcr$ or $P?cr$ then $\vdash_L p$ implies that p is INM-valid.

Proof: The proof is based upon induction on the length of the "proof" of p :

If the proof is a single step then either p is a propositional tautology (and hence valid by definition) or p is an axiom (and then p is valid according to theorems 3.2.2, and 3.2.4 through 3.2.7 (and, as required 3.2.8, 3.2.9 and 3.2.10)).

If the proof of p is longer than one step, then p is arrived at by the use of either MP or RN. If p is proved from q and $q \rightarrow p$ by MP then $\vdash_L q$ and $\vdash_L q \rightarrow p$ appear as earlier steps of the proof. By the induction hypothesis q and $q \rightarrow p$ are INM-valid. Therefore p is INM-valid by 3.2.1. If $p = [a]q$ and q appears as an earlier step in the proof of p then q is INM-valid by the induction hypothesis. Therefore p is also INM-valid by 3.2.3. ■

The soundness of our PDL with respect to the standard Kripke models is an immediate corollary.

3.2.12 Corollary: If L is one of $P, P?, Pc, Pr, P?c, P?r, Pcr$ or $P?cr$ then: If $\vdash_L p$ then

p is SKM-valid.

Proof: $SKM \subset INM$ (theorem 3.1.3). ■

Notice however that none of our list of PDL is sound with respect to non-KM (or, therefore KM). The reason, of course, is that instances of $Ax.Ind$ are not necessarily true in non-standard models. In other words,

3.2.13 Theorem: There are non-standard models which are not induction models, i.e., $INM \subset KM$.

Proof: Consider the following non-standard model $\mathfrak{S}=(W,\pi,\nu)$ where

$$\begin{aligned} W &= \{u, v\}, \\ \pi(a) &= \{(u, u), (v, v)\} = \pi(\lambda), \\ \pi(a^*) &= \{(u, u), (v, v), (u, v)\}, \\ \nu(p) &= \{u\}. \end{aligned}$$

Obviously, $u \in \nu(p)$ and $u \in \nu([a^*](p \rightarrow [a]p))$ (the latter is a tautology after expanding by 3.1.1). However, if instances of IN are true, then also $u \in \nu([a^*]p)$, which, by 3.1.1e, means that $w \in \nu(p)$ - a contradiction. ■ {NOTE: 3.2.13 proof}

The following is a standard corollary of the soundness theorem: consistency follows from (local) satisfiability.

3.2.14 Corollary: If INM is non-empty then P , $P?$, Pc , Pr , $P?c$, $P?r$, Pcr and $P?cr$ are consistent.

Proof: $\perp$ is not locally true in any INM model (3.1.1a), thus by 3.2.12, not $\vdash \perp$. ■

Before preceding to the next section, let us consider the soundness of the logic D and its offspring. Let DKM (SDKM, non-SDKM, DINM) be the class of Kripke models (standard Kripke models, non-standard Kripke models, induction models) which are "serial". Specifically, for $\mathfrak{S} = (W, \pi, \nu)$ and $\mathfrak{S} \in KM$ (SKM, non-SKM, INM), $\mathfrak{S} \in DKM$ (SDKM, non-SDKM, DINM) if and only if for all $a \in \Pi$,

$$(\text{serial}) \quad \forall v \in W, \exists w \in W \text{ such that } (v, w) \in \pi(a).$$

3.2.15 Theorem: Let $a \in \Pi$. For any model $\nu = (W, \pi, \nu)$ which satisfies the serial condition, if $\mathfrak{S} \models p$ then $\mathfrak{S} \models \langle a \rangle p$ (i.e., RP preserves validity in $\mathfrak{S}$).

Proof: Suppose that not $\mathfrak{S} \models \langle a \rangle p$. Therefore, by 3.1.1d, for all $v \in W$, either $(w, v) \notin \pi(a)$ or not $\mathfrak{S} \models_v p$. Since π satisfies the serial condition, it must be that, for all $v \in W$ (including w), not $\mathfrak{S} \models_v p$. ■

3.2.16 The soundness theorem for logics D, D?, Dc, Dr, D?c, D?r, Dcr and D?cr: If L is one of D, D?, Dc, Dr, D?c, D?r, Dcr or D?cr then $\vdash_L p$ implies that p is DINM (SDKM)-valid.

Proof: As in proof 3.2.11 (3.2.12) with the addition of 3.2.15. ■

3.2.17 Corollary: If DINM is non-empty then D is consistent.

Proof: $\perp$ is not locally true in any INM model (3.1.1a), thus by 3.2.16, not $\vdash \perp$. ■

We shall see in the next chapter (section 4.1) that INM and DINM are indeed non-empty, thus ensuring the consistency of all of the logics under our consideration.

3.3 Semantic Compactness and Semantic Consequence

In classical logic, a set of models is said to be *compact* whenever the satisfiability of a set of formulae corresponds to the satisfiability of every finite subset of it. We would like our semantics to be compact "... since in practical applications, we can only make finite descriptions of situations in which we want to cover an infinite number of cases." [BER79; pp.163]

The presence of two types of satisfiability in modal and dynamic logics bring about a dichotomy in the notion of compactness. A class of models, say Ω , is said to be locally compact if every set of formulae Γ can be simultaneously (i.e., at the same state) Ω -satisfied when and only when every finite subset Γ' of Γ can be simultaneously Ω -satisfied. Ω is said to be globally compact if every set of formulae Γ can be globally

Ω -satisfied when and only when every finite subset Γ' of Γ can be globally Ω -satisfied. Obviously, global compactness implies local compactness but not vice versa (since there cannot be any general method of going from the local to the global variety of compactness).

3.3.1 Theorem: SKM is not locally compact.

Proof: Consider the following counter example to local compactness. Given formula $P \in \Sigma_0$ and program $a \in \Pi$ consider the infinite set $\Gamma = \{[a^n]P \mid n < \omega\} \cup \{[a^*]P\}$. Let $N_n = (N, R, \nu_n)$ be the standard Kripke model where

- a) N is the set of natural numbers,
- b) $(n, m) \in R(a)$ iff $m = n + 1$ (i.e., $R(a)$ is the relation of immediate successor),
- c) $\nu_n(P) = \{i \mid i \leq n\}$ (i.e., valuation depends only upon $n \in N$).

For any nonempty finite subset Γ' of Γ there is a $j \in N$ such that $\{[a^*]P\} \cup \{[a^i]P \mid i \leq j\} \supseteq \Gamma'$, but, for any $k \in N$, $[a]^{j+k+1}P \notin \Gamma'$. Thus, any finite subset of Γ can be locally (in fact, globally) SKM-satisfied. However, according to the $*$ -ancestral constraint on SKM, the formula $[a^*]P$ means that there must be some $k < \omega$ such that $[a^k]P$ is locally SKM-satisfied, which contradicts $\Gamma \not\subseteq \{[a^n]P \mid n < \omega\}$. Therefore, SKM is not locally compact. (NOTE: 3.3.1 proof) ■

3.3.2 Corollary: SKM is not globally compact.

Proof: Since SKM is not locally compact, it also cannot be globally compact. ■

Notice, that the example used in the proof of 3.3.1 cannot be applied against KM since the strict $*$ -ancestral condition is required. In fact, KM and INM are locally compact. (NOTE: proof of compactness) If this seems counter intuitive, note that KM and SKM compactness are based on distinct notions of KM and SKM-satisfaction, respectively.

We continue with our policy of splitting semantical concepts, now dealing with the notion of logical consequence. Once again the distinction is based on the two distinct concepts of truth (i.e., true in at a state within a model and globally true in a model).

Let Ω be a set of models. We say that a formula p is a Ω -(local) consequence of a set of formulae Γ , and write $\Omega, \Gamma \models p$, if and only if, for all models $\mathcal{S} = (W, \pi, \nu) \in \Omega$ and for all $w \in W$, $\mathcal{S} \models_w \Gamma$ implies that $\mathcal{S} \models_w p$. We say that a formula p is a Ω -global consequence of a set of formulae Γ and write $\Omega, \Gamma \models^g p$ if and only if, for all models $\mathcal{S} = (W, \pi, \nu) \in \Omega$, if $\mathcal{S} \models \Gamma$ then $\mathcal{S} \models p$.

Notice that whenever $\Gamma = \emptyset$ then both of these types of semantic consequence actually coincide with Ω -validity. However, when Γ is non-empty, local and global consequence can be very different. This can be demonstrated by the following simple example. Although $\text{SKM}, \{p\} \models^g [a]p$ (by 3.2.3), it is not the case that $\text{SKM}, \{p\} \models [a]p$ since for any standard Kripke model $\mathcal{S} = (W, \pi, \nu)$, $\pi(a)$ is not necessarily a reflexive relation.

It is quite clear, furthermore, that local consequence is a stronger condition than global consequence. This is the point of the next theorem.

3.3.3 Theorem: If $\Omega, \Gamma \models p$ then $\Omega, \Gamma \models^g p$.

Proof: Let $\mathcal{S} = (W, \pi, \nu) \in \Omega$ and suppose that $\mathcal{S} \models \Gamma$. Since $\mathcal{S} \models_w \Gamma$ implies $\mathcal{S} \models_w p$ for all $w \in W$ then $\mathcal{S} \models p$. ■

The definition of compactness used above motivates the following. A class Ω of models is said to be compact with respect to global consequence if Ω has the following property:

for all $p \in \Sigma$ and $\Gamma \subseteq \Sigma$, $\Omega, \Gamma \models^g p$ iff there is some finite subset $\Gamma' \subseteq \Gamma$ such that $\Omega, \Gamma' \models^g p$.

A class Ω of models is said to be compact with respect to local consequence if Ω has the following property:

for all $p \in \Sigma$ and $\Gamma \subseteq \Sigma$, $\Omega, \Gamma \models p$ iff there is some finite subset $\Gamma' \subseteq \Gamma$ such that $\Omega, \Gamma' \models p$.

Actually, global and local compactness coincide with compactness with respect to global and local consequence respectively. This is shown in the next two theorems.

3.3.4 Theorem: A class of models Ω is globally compact iff Ω is compact with respect to global consequence.

Proof: First assume that Ω is globally compact: (A) Let $\Omega, \Gamma \models^g p$. Γ must be globally Ω -satisfiable (for otherwise, by the global compactness of Ω , Γ has a finite subset, say Γ' , which is also not globally Ω -satisfiable, so that $\Omega, \Gamma' \not\models p$ - vacuously). Now suppose that there is no finite subset Γ' of Γ such that $\Omega, \Gamma' \models^g p$, i.e., for any finite $\Gamma' \subseteq \Gamma$, there is a $\mathfrak{S} \in \Omega$ such that $\mathfrak{S} \models \Gamma'$ but not $\mathfrak{S} \models p$. Therefore, for every finite $\Gamma' \subseteq \Gamma$, $\Gamma' \cup \{p\}$ is not globally Ω -satisfiable. By the global compactness of Ω , it follows that $\Gamma \cup \{p\}$ is not globally Ω -satisfiable. Since $\Omega, \Gamma \models^g p$ then Γ cannot be globally Ω -satisfiable, which contradicts the earlier result. Thus, there is a finite $\Gamma' \subseteq \Gamma$ such that $\Omega, \Gamma' \models^g p$. (B) Now suppose that there is a finite subset $\Gamma' \subseteq \Gamma$ such that $\Omega, \Gamma' \models^g p$. For $\mathfrak{S} \in \Omega$, if $\mathfrak{S} \models \Gamma$ then $\mathfrak{S} \models \Gamma'$ so that $\mathfrak{S} \models p$ also. By (A) and (B) we have that Ω is globally compact with respect to global consequence.

Conversely, assume that Ω is compact with respect to global consequence. (C) Let $\mathfrak{S} \in \Omega$ and suppose that $\mathfrak{S} \models \Gamma$, then any finite $\Gamma' \subseteq \Gamma$ is also globally satisfied by $\mathfrak{S}$. (D) Now suppose that every finite $\Gamma' \subseteq \Gamma$ is globally Ω -satisfiable but that Γ itself is not (i.e., given any $\mathfrak{S} = (W, \pi, \nu) \in \Omega$, we can find $p \in \Gamma$ and $w \in W$ such that $\mathfrak{S} \not\models p$). Let $p \in \Gamma$. Now $\Omega, \Gamma \models^g \neg p$ (vacuously). Therefore, there is some finite $\Gamma' \subseteq \Gamma$ such that $\Omega, \Gamma' \models^g \neg p$. Since $\Gamma' \cup \{p\}$ is globally Ω -satisfiable, say by $\mathfrak{S}$, then $\mathfrak{S} \models \Gamma'$. Consequently, $\mathfrak{S} \models \neg p$, and therefore, since $\mathfrak{S} \models p$, $\mathfrak{S} \models \neg p \& p$ or $\mathfrak{S} \models \perp$, which directly contradicts 3.1.1a. Thus, it must be that Γ is globally Ω -satisfiable. By (C) and (D) we have that Ω is globally compact. ■

The proof of theorem 3.3.5 is almost a duplication of the form of 3.3.4. The details, however, are a little different so we give it here in full.

3.3.5 Theorem: A class of models Ω is locally compact iff Ω is compact with respect to local consequence.

Proof: First assume that Ω is locally compact: (A) Assume there is no finite $\Gamma' \subseteq \Gamma$ with the property $\Omega, \Gamma' \models p$, i.e., there is a $\mathfrak{S} = (W, \pi, \nu) \in \Omega$ and $w \in W$ such that $\mathfrak{S} \models \Gamma'$ but

$\mathfrak{S} \models_w \neg p$. Therefore, for every finite $\Gamma' \subseteq \Gamma$, $\Gamma' \cup \{\neg p\}$ is Ω -satisfiable, and so, by the local compactness of Ω , $\Gamma \cup \{\neg p\}$ is Ω -satisfiable. (B) Now suppose that there is a finite subset $\Gamma' \subseteq \Gamma$ such that $\Omega, \Gamma' \models p$. For $\mathfrak{S} = (W, \pi, \nu) \in \Omega$, if $\mathfrak{S} \models_w \Gamma$ then $\mathfrak{S} \models_w \Gamma'$ so that $\mathfrak{S} \models_w p$ also. By (A) and (B) we have that Ω is locally compact with respect to local consequence.

For the converse, assume that Ω is compact with respect to local consequence. (C) Let $\mathfrak{S} = (W, \pi, \nu) \in \Omega$ and suppose that for all $w \in W$, $\mathfrak{S} \models_w \Gamma$, then any finite $\Gamma' \subseteq \Gamma$ is also satisfied by $\mathfrak{S}$ at w . (D) Now suppose that every finite $\Gamma' \subseteq \Gamma$ is Ω -satisfiable but that Γ itself is not (i.e., given any $\mathfrak{S} = (W, \pi, \nu) \in \Omega$, we can find $p \in \Gamma$ and $w \in W$ such that $\mathfrak{S} \not\models_w p$). Now $\Omega, \Gamma \not\models p$ for any $p \in \Gamma$ (vacuously). Therefore, there is some finite $\Gamma' \subseteq \Gamma$ such that $\Omega, \Gamma' \not\models p$. Since $\Gamma' \cup \{p\} \subseteq \Gamma$ is Ω -satisfiable, say by $\mathfrak{S}$, then $\mathfrak{S} \models \Gamma'$. Consequently, $\mathfrak{S} \models \neg p \& p$ (by 3.1.1f) or $\mathfrak{S} \models \perp$, which directly contradicts 3.1.1a. Thus, it must be that Γ is Ω -satisfiable. By (C) and (D) we have that Ω is locally compact. ■

Accordingly, both versions of semantical consequence fail to be compact.

In the absence of compactness, as in the context of standard Kripke models, the problem of deciding whether a given formula is a consequence (global or local) of other formulae is of considerable practical interest. Unfortunately, this problem is highly undecidable. (NOTE: complexity of the consequence problem)

3.4 Collapsed and Finite Model Properties

The process of "filtration" of modal logic takes a given model and partially constructs others of a "manageable" size which preserves the truth value of a selected set of sentences. This technique has been used successfully to demonstrate that many modal logics have the finite model property: any satisfiable formula has a finite model which satisfies it. Once the finite model property for a logic has been affirmed then its decidability usually follows. In section 3.1 we showed that all satisfiable formulae are satisfiable by a countable model and that, when the *repeat* operator is absent then,

this formula is satisfiable by a finite model. In this section we strengthen this result considerably by actually constructing a finite model for a satisfiable $\Sigma_{PDL, \tau, c}$ formula. In chapter four (section 4.2) we extend the "filtration" method of modal logic so that it can be applied to PDL. The extended filtration process is called *Fischer-Ladner filtration*. (NOTE: chapter 4) A related idea, which is just a special case of Fischer-Ladner filtration, is introduced here in order to discuss a special feature of some PDL, called the collapsed model property. As in the case of filtration, the construction used in this section takes a model and yields another of a finite size. The resulting model preserves the truth value of a distinguished set of sentences. Unlike the filtration technique, which fixes a family of models each of which is only partially defined, this procedure fully constructs a single model, which we call the *quotient model*.

Let $F \subseteq \Sigma$ and let $\mathfrak{S} = (W, \pi, \nu)$ be any model. F induces an equivalence relation $\simeq_F$ on W defined as follows:

$$u \simeq_F v \text{ if and only if for all } q \in F (\mathfrak{S} \models_u q \text{ iff } \mathfrak{S} \models_v q).$$

- Let $[u]_F$ denote the equivalence class of u modulo $\simeq_F$ (i.e., $[u]_F = \{v \mid v \simeq_F u\}$). (We will usually drop the subscript whenever there is no chance of ambiguity.) A F -Quotient (or F -Factor) model of $\mathfrak{S}$ is a model $\mathfrak{S}/F = (W/F, \pi/F, \nu/F)$ where:

- W/F is the partition of W induced by $\simeq_F$ (i.e., W/F is the set of all equivalence classes modulo $\simeq_F$, called *quotient states*),
- for all $a \in \Pi$, $\pi/F(a) = \{([u], [v]) \mid (u, v) \in \pi(a)\}$,
- for all $P \in \Sigma_0$, $\nu/F(P) = \{[u] \mid u \in \nu(P)\}$.

Note that π/F is well defined - i.e., given that $u \simeq u'$, $v \simeq v'$ and that $(u, v) \in \pi(a)$ and $([u], [v]) \in \pi/F(a)$, it is clear that $([u'], [v']) \in \pi/F(a)$ since $[u] = [u']$ and $[v] = [v']$. Note also, that if F has n members then W/F has at most 2^n . The next theorem shows that quotient models retain the important KM, SKM and non-SKM characteristics of their progenitor models. This is also true when the progenitor model is from INM (3.4.3).

3.4.1 Theorem: Let $\mathfrak{S}=(W,\pi,\nu)$ and let $F \subseteq \Sigma_{\text{PDL},\gamma,c}$.

If $\mathfrak{S} \in \text{SKM}$ then $\mathfrak{S}/F \in \text{SKM}$;

If $\mathfrak{S} \in \text{non-SKM}$ then $\mathfrak{S}/F \in \text{non-SKM}$;

If $\mathfrak{S} \in \text{KM}$ then $\mathfrak{S}/F \in \text{KM}$.

Proof: We are only required to show that the union, composition, λ -identity, test, inverse, and $*$ -ancestral and $*$ -non-standard (and therefore $*$ -reflexive, $*$ -inherit and $*$ -transitive) conditions of π/F are inherited from π .

union: Assume π satisfies the union condition. Then $(|u|,|v|) \in \pi/F(a+b)$ iff $(u,v) \in \pi(a+b)$ iff $(u,v) \in \pi(a) \cup \pi(b)$ iff $(|u|,|v|) \in \pi/F(a) \cup \pi/F(b)$.

composition: Assume π satisfies the composition condition. Then $(|u|,|v|) \in \pi/F(ab)$ iff $(u,v) \in \pi(ab)$ iff $(u,v) \in \pi(a) \circ \pi(b)$ iff $(|u|,|v|) \in \pi/F(a) \circ \pi/F(b)$.

λ -identity: Assume π satisfies the λ -identity condition. Then $\pi/F(\lambda) = \{(|u|,|u|) \mid u \in W\}$ by definition.

test: Assume π satisfies the test condition. Then, by definition, $\pi/F(p) = \{(|u|,|u|) \mid u \in \nu(p)\}$, and since $u \in \nu(p)$ iff $|u| \in \nu/F(p)$, π/F satisfies the test condition.

inverse: Assume π satisfies the inverse condition. Then $(|v|,|u|) \in \pi/F(a^c)$ iff $(v,u) \in \pi(a^c)$ iff $(u,v) \in \pi(a)$ iff $(|u|,|v|) \in \pi/F(a)$.

$$ -ancestral:* Assume π satisfies the $*$ -ancestral condition. Then $(|u|,|v|) \in \pi/F(a^*)$ iff $(u,v) \in \pi(a^*)$ iff $\exists n < \omega$ such that $(u,v) \in \pi(a)^n$ iff $\exists n < \omega$ such that $(|u|,|v|) \in \pi/F(a)^n$.

$$ -non-standard:* Assume π satisfies the $*$ -non-standard condition. Let $(|u|,|v|) \in \pi/F(a)^n$ for some $n < \omega$. Then $(u,v) \in \pi(a)^n$ and so $(u,v) \in \pi(a^*)$ whence $(|u|,|v|) \in \pi/F(a^*)$.

The extension of the notion of subformula (chapter two, section 2.1) to deal with the myriad collection of dynamic modalities was first introduced by Fischer and Ladner in [F&L77] for Σ_{PDL} and [F&L79] for $\Sigma_{\text{PDL},\gamma}$. It is critical in the proof of completeness (chapter four) and decidability (section 3.5). Our version also incorporates converse and repeat. (NOTE: FLC extension) The Fischer-Ladner closure of a set Γ of formulae, denoted by $\text{FLC}(\Gamma)$, is the smallest set such that:

- a) $\Gamma \subseteq \text{FLC}(\Gamma)$,
- b) if $p \in \Gamma$ then $\text{SUBF}(p) \subseteq \text{FLC}(\Gamma)$ i.e., $\text{FLC}(\Gamma)$ is closed under subformulae,
- c) if $\langle a+b \rangle p \in \text{FLC}(\Gamma)$ then both $\langle a \rangle p, \langle b \rangle p \in \text{FLC}(\Gamma)$,
- d) if $\langle ab \rangle p \in \text{FLC}(\Gamma)$ then $\langle a \rangle \langle b \rangle p \in \text{FLC}(\Gamma)$,
- e) if $\langle a^* \rangle p \in \text{FLC}(\Gamma)$ then $\langle a \rangle \langle a^* \rangle p \in \text{FLC}(\Gamma)$,
- f) if $\langle q? \rangle p \in \text{FLC}(\Gamma)$ then both $p, q \in \text{FLC}(\Gamma)$,
- g) if $\langle a^c \rangle p \in \text{FLC}(\Gamma)$ then $\langle a \rangle \neg \langle a^c \rangle p \in \text{FLC}(\Gamma)$,
- h) if $\text{repeat}(a) \in \text{FLC}(\Gamma)$ then $\langle a \rangle \text{repeat}(a) \in \text{FLC}(\Gamma)$.

We then say that a set Γ is closed under the Fischer-Ladner conditions if $\Gamma = \text{FLC}(\Gamma)$. Unlike the notion of subformulae, the Fischer-Ladner closure is more than a simple "decomposition". It is useful (i.e., it works - it precipitates finite models that yield decidability and completeness results with respect to SKM and INM) because it somehow captures "all the formulae that are relevant to the semantics of $[\Gamma]$ " [PAR81;pp.8]. But why it should do this, especially in the case of converse (rule g), is somewhat mysterious!

Although any formula is composed of a finite number of atomic formulae and operators it is not obvious that if Γ is a finite set of formulae, then $\text{FLC}(\Gamma)$ is also finite. Indeed rules d),e),g) and h) actually increase in "length". However, it turns out that these rules do not increase the size of $\text{FLC}(\Gamma)$ by any more than the others. In fact, the size or cardinality of $\text{FLC}(\Gamma)$ is linear in the sum of the "lengths" of all the formulae of Γ . This is the Fischer-Ladner lemma which we now demonstrate. (NOTE: proof of Fischer-Ladner lemma)

Let $\text{len}(p)$ and $\text{len}(\Gamma)$ denote the number of (tokenwise) occurrences of symbols in $p \in \Sigma$ and in all $q \in \Gamma \subseteq \Sigma$ respectively (we count *repeat* as a single symbol). That is, $\text{len}(p)$ is the length of p regarded as a string over the alphabet $\Sigma_0 \cup \Pi_0 \cup \text{OpForm} \cup \text{OpProg} \cup \{ (,) \}$. Also let $\text{card}(\Gamma)$ be the cardinality of the set Γ .

3.4.2 The Fischer-Ladner Lemma: If Γ is a finite set of formulae, then $\text{FLC}(\Gamma)$ is also

finite and, in fact, $\text{card}(\text{FLC}(\Gamma)) = \text{len}(\Gamma)$.

Proof: The proof is by induction on a modified length of the formulae of Γ - but first, some observations. First note that, for any $\Gamma_1, \Gamma_2 \subseteq \Sigma$, $\text{FLC}(\Gamma_1 \cup \Gamma_2) = \text{FLC}(\Gamma_1) \cup \text{FLC}(\Gamma_2)$. And so, it follows that $\text{FLC}(\Gamma) = \bigcup_{p \in \text{FLC}(\Gamma)} \text{FLC}(\{p\})$. Therefore, since Γ is finite,

$$\text{card}(\text{FLC}(\Gamma)) \leq \sum_{p \in \text{FLC}(\Gamma)} \text{card}(\text{FLC}(\{p\})). \quad (1)$$

Now, for $p \in \text{FLC}(\Gamma)$, consider the following set: Let $\text{CL}(p)$ be the smallest set such that

- a') $p \in \text{CL}(p)$,
- bb) if $\neg q \in \text{CL}(p)$ then $q \in \text{CL}(p)$,
- bc) if $qvr \in \text{CL}(p)$ then $q, r \in \text{CL}(p)$,
- bd) if $\langle A \rangle q \in \text{CL}(p)$ then $q \in \text{CL}(p)$, $A \in \Sigma_0 \cup \{\lambda\}$,
- c') if $\langle a+b \rangle q \in \text{CL}(p)$ then $\langle a \rangle \psi(q), \langle b \rangle \psi(q), q \in \text{CL}(p)$,
- d') if $\langle a;b \rangle q \in \text{CL}(p)$ then $\langle a \rangle \psi(\langle b \rangle q), \langle b \rangle q \in \text{CL}(p)$,
- e') if $\langle a^* \rangle q \in \text{CL}(p)$ then $\langle a \rangle \psi(\langle a^* \rangle q), q \in \text{CL}(p)$,
- f') if $\langle r? \rangle q \in \text{CL}(p)$ then $r, q \in \text{CL}(p)$,
- g') if $\langle a^c \rangle q \in \text{CL}(p)$ then $\langle a \rangle \neg \psi(\langle a^c \rangle q), q \in \text{CL}(p)$,
- h') if $\text{repeat}(a) \in \text{CL}(p)$ then $\langle a \rangle \psi(\text{repeat}(a)) \in \text{CL}(p)$.

Strings of the form $\psi(- -)$ are called ψ -variables. For $q \in \text{CL}(p)$ let $\gamma(q)$ be the same as $\text{len}(p)$ but where token occurrences of symbols from $\{(), \neg, \psi\}$ and those bound by ψ -variables are not counted, and the converse symbol is counted twice. (Remember the *repeat* operator is counted as one symbol). For example, if q is the string $\langle A^c; B \rangle \psi(\langle b \rangle \text{repeat}(a))$ where $A, B \in \Pi_0$ then $\gamma(q) = 5$. Clearly, we have

$$\gamma(q) \leq \text{len}(q). \quad (2)$$

Let Ψ be the set of ψ variables introduced by completing $\text{CL}(p)$. Thus, taking $q \in \text{CL}(p)$ as in the above example, $\psi(\langle b \rangle \text{repeat}(a)) \in \Psi$. $\text{CL}(p)$ is defined so that

- (i) only one rule can be applied to a specified member of $\text{CL}(p)$;
- (ii) if $\psi(q) \in \text{CL}(p)$ then $q \in \text{CL}(p)$ - and q cannot be counted by γ more than once;
- (iii) if $\langle a \rangle q \in \text{CL}(p)$ then $q \in \text{CL}(p)$.

(i) and (iii) are transparent. (ii) is confirmed by the observation that $\psi(q)$ is only introduced by a rule whenever q already occurs as the premise or also appears in the

2

of/de

2

MicroD

list of consequences (not bound by ψ). (ii) and (iii) combine to give

$$\text{card}(\text{FLC}(\{p\})) = \text{card}(\text{CL}(p) - \Psi). \quad (3)$$

We now prove, by induction on $\gamma(p)$, that

$$\text{card}(\text{CL}(p) - \Psi) \leq \gamma(p). \quad (4)$$

When $\gamma(p)=1$ then $p \in \Sigma_0$, $\Psi = \emptyset$, $\text{CL}(p) = \{p\}$ and therefore $\text{card}(\text{CL}(p) - \Psi) = 1$.

Now suppose that $\gamma(p) = n+1$. Let ρ denote the $\text{CL}(p)$ rule to be applied to p . Notice that ρ (as applied to p) is of the form

$$p \in \text{CL}(p) \text{ implies } p_1, p_2, \dots, p_k \in \text{CL}(\{p\}).$$

Since ρ is the only rule applicable to p it follows that

$$\text{CL}(p) = \{p\} \cup (\cup_{i=1}^k \text{CL}(p_i)). \quad (5)$$

Observe, also, that

$$\gamma(p) = \gamma(p_1) + \gamma(p_2) + \dots + \gamma(p_k) + 1. \quad (6)$$

Therefore,

$$\text{card}(\text{CL}(p) - \Psi) = \text{card}(\{p\} \cup (\cup_{i=1}^k \text{CL}(p_i)) - \Psi) \quad \text{--- (by (5))}$$

$$\text{card}(\text{CL}(p) - \Psi) = 1 + \sum_{i=1}^k \text{card}(\text{CL}(p_i) - \Psi)$$

$$\text{card}(\text{CL}(p) - \Psi) \leq 1 + \sum_{i=1}^k \gamma(p_i) \quad \text{(by the induction hypothesis)}$$

$$\text{card}(\text{CL}(p) - \Psi) \leq \gamma(p). \quad \text{(by (6))}$$

(2),(3) and (4) combine to give

$$\text{card}(\text{FLC}(\{p\})) \leq \text{len}(p),$$

and therefore, by (1)

$$\text{card}(\text{FLC}(\Gamma)) \leq \text{len}(\Gamma).$$

■

Let L be a PDL with well-formed formulae Σ and let Ω be a class of models. We say that L has the collapsed Ω -model property if for any formulae $p \in \Sigma$ and any model $\mathfrak{S} = (W, \pi, \nu) \in \Omega$

a) $\mathfrak{S} / \text{FLC}(\{p\}) \in \Omega$;

b) if p is Ω -locally satisfied by $\mathfrak{S}$ (i.e., $\mathfrak{S} \models_x p$ for some $x \in W$) then, for any $w \in W$ and $q \in \text{FLC}(\{p\})$, $\mathfrak{S} \models_w q$ if and only if $\mathfrak{S} / \text{FLC}(\{p\}) \models_{[w]} q$.

Condition a) is guaranteed by 3.4.1 when Ω is one of **KM**, **SKM** or **non-SKM**.

Theorem 3.4.4 guarantees condition a) when Ω is INM. Condition b) states that $\mathcal{S}$ and $\mathcal{S}/\text{FLC}(\{C\})$ must agree upon which formulae of $\text{FLC}(\{C\})$ are true and which are false. The next theorem ensures that the PDLs P , $P?$, Pc and $P?c$ all enjoy the collapsed Ω -model property.

3.4.3 Theorem: Let $p \in \Sigma_{\text{PDL},?,c}$ and $\mathcal{S} = (W, \pi, \nu) \in \Omega$ be a model which Ω -locally satisfies p . For any $w \in W$ and $q \in \text{FLC}(\{C\})$, $\mathcal{S} \models_w q$ if and only if $\mathcal{S}/\text{FLC}(\{p\}) \models_w q$.

Proof: Assume that $q \in F = \text{FLC}(\{p\})$. The proof is based upon induction on the complexity of q :

Case 1- $q = P \in \Sigma_0$: According to the definition of $\mathcal{S}/F$, $w \in \nu(P)$ iff $|w| \in \nu/F(P)$.

Case 2- $q = \neg r$: $\mathcal{S} \models_w \neg r$ iff not $\mathcal{S} \models_w r$ (by 3.1.1b) iff not $\mathcal{S}/F \models_w r$ (by the induction hypothesis) iff $\mathcal{S}/F \models_w \neg r$ (by 3.1.1b).

Case 3- $q = \text{svr}$: $\mathcal{S} \models_w \text{svr}$ iff $\mathcal{S} \models_w s$ or $\mathcal{S} \models_w r$ (by 3.1.1c) iff $\mathcal{S}/F \models_w s$ or $\mathcal{S}/F \models_w r$ (by the induction hypothesis) iff $\mathcal{S}/F \models_w \text{svr}$ (by 3.1.1c).

Case 4- $q = \langle a \rangle r$: $\mathcal{S} \models_w \langle a \rangle r$ iff $(w, v) \in \pi(a)$ for some $v \in W$, and $\mathcal{S} \models_v r$ (by 3.1.1d) iff $(|w|, |v|) \in \pi/F(a)$ for some $|v| \in W/F$, and $\mathcal{S} \models_{|v|} r$ (by the induction hypothesis) iff $\mathcal{S}/F \models_{|w|} \langle a \rangle r$ (by 3.1.1d). ■

3.4.4 Corollary: Let $\mathcal{S} = (W, \pi, \nu) \in \text{INM}$ and let $F \subseteq \Sigma_{\text{PDL},?,c}$, then $\mathcal{S}/F \in \text{INM}$.

3.4.5 Corollary: PDL P , $P?$, Pc and $P?c$ have the collapsed KM, SKM, non-SKM and INM-model property.

Proof: 3.4.1 (3.4.4) and 3.4.3 give condition a) and b) of the collapsed model property respectively. ■

Logics that support the *repeat* operator (i.e., the PDL P_r , $P?_r$, Pc_r and $P?c_r$) on the other hand, lack the collapsed model property. The example used to demonstrate 3.4.6 is a standard Kripke model - but no use is made of this fact in the proof. Hence corollary 3.4.7.

3.4.6 Theorem: The PDL P_r , $P?_r$, Pc_r and $P?c_r$ fail to have the collapsed SKM-model property.

Proof: Consider the following SKM model $\mathfrak{S} = (U, \rho, \tau)$ where

$$U = \{u_n\}_{n \geq 0}$$

$$(u_i, u_j) \in \rho(a) \text{ iff } i = j+1.$$

$\mathfrak{S}$ is composed of infinitely many downward transitions between consecutive states - thus $\mathfrak{S}$ locally satisfies $\langle a^n \rangle \top$ at u_n for any $n < \omega$. However, $\neg \text{repeat}(a)$ is globally satisfied. Now, any FLC(Γ)-quotient version of $\mathfrak{S}$ based on finite $F = \text{FLC}(\Gamma)$ must be finite (i.e., U/F is of size $2^{\text{card}(F)}$). Consequently, there are distinct states u_k and u_m , say, where $k < m$ such that $|u_k| = |u_m|$. Now since $(|u_m|, |u_{m-1}|) \in \pi/F(a)$, $\dots$, $(|u_{k+1}|, |u_k|) \in \pi/F(a)$ then $\text{repeat}(a)$ is true at any of the quotient states $|u_k|, \dots, |u_m|$. Thus, part b of the collapsed SKM-model property is violated. (NOTE: Pratt's example)

3.4.7 Corollary: The PDL Pr , P?r , Pcr , and P?cr fail to have the collapsed INM, KM or non-SKM-model property.

Let L be a PDL with well-formed formulae Σ and let Ω be a class of models. We say that L has the finite Ω -model property if for any Ω -satisfiable formulae $p \in \Sigma$ there is a finite model $\mathfrak{S} = (W, \pi, \nu) \in \Omega$ satisfying p . Obviously, if L has the collapsed Ω -model property then it has the finite Ω -model property. Thus, we have two independent proofs (the other being a straight forward corollary of 3.1.4/5) that PDL P , P? , Pc and P?c have the finite Ω -model property.

Despite the fact that Pr and P?r lack the collapsed Ω -model property, they can be shown to have the finite Ω -model property. The proof of this fact will not be undertaken in this thesis because it relies too heavily on pure automaton theory. Briefly, the proof is as follows: Ω models of Pr or P?r formulae are viewed as labelled graphs. Such a labelled graph may be "unravelled" or "unwound" into a tree structure where programs connect only descendent nodes (this is the *tree model property* - discussed further in the next section). It is then shown that there is a finite automaton which recognizes these tree structures. Thus, by a result due to M.O. Rabin (NOTE: Rabin), it follows that the Ω -satisfiable formula in question has a finitely generable tree which has a corresponding finite graph - and this finite graph is but a

representation of a finite Ω -model. Thus, every Ω -satisfiable formula of Pr or P?r has a finite model. (NOTE: finite models for *repeat*) Logics Pcr and P?cr however, lack even the finite model property. In these cases the converse and repeat operators combine to destroy the possibility of finite models.

3.4.8 Theorem: PDL Pcr and P?cr do not have the finite model property.

Proof: Consider the $\Sigma_{\text{PDL,c,repeat}}$ formula

$$p = \text{repeat}(a) \ \& \ \neg \langle a^* \rangle \text{repeat}(a^c).$$

p is globally satisfied by the model $N_n = (N, R, \nu_n)$ defined in the proof of 3.3.1, and so p is satisfiable. Now suppose that $\mathfrak{S} = (W, \pi, \nu)$ is any model and that $u_0 \in \nu(p)$. Then $u_0 \in \nu(\text{repeat}(a))$ and so there are $u_1, u_2, \dots$, etc. such that $(u_0, u_1) \in \pi(a)$, $(u_1, u_2) \in \pi(a)$, $\dots$, etc. If for any $i < j$, $u_i = u_j$, then $u_i \in \nu(\text{repeat}(a^c))$, and so we have a contradiction. Therefore each of u_i are distinct, which implies that W must be infinite. ■ (NOTE: Streett's example)

3.5 Decidability and Complexity

In this section we briefly turn our attention to the issue decidability. A problem is decidable if it has an effective and finitary solution. Obviously, there are two problems for any of the PDL under our consideration which directly concern us. The first of these is to decide whether a particular formula is a theorem of a logic or not. The second problem is to tell whether or not a formula is valid. Let L be one of all of P , P? , Pc , Pr , P?c , P?r , Pcr or P?cr and let Ω be one of KM , SKM , non-SKM or INM .

The positive test of theoremhood in L is straightforward: we know that proofs of all the theorems of L can be effectively enumerated since a formula appears in a proof only if it is a member of our decidable list of axioms, or it is inferred from an earlier formula (that also appears in the proof) by the rules RN or MP . We can eventually verify that a particular formula is a theorem by simply inspecting our "list" of proofs of L .

Moreover, if L is sound (with respect to Ω) then, if we can determine a model $\mathfrak{S} \in \Omega$ for which p is not globally Ω -satisfiable then p is not a theorem of L . Therefore, if L had the finite Ω -model property then we automatically have a way of deciding whether a formula is a non-theorem: The idea here is to find a falsifying model for the formula in question. We can enumerate all the finite π -unrestricted models (modulo isomorphism), say $\mathfrak{C}_1, \mathfrak{C}_2, \dots$, etc. If p is not a theorem of L then for some n , not $\mathfrak{C}_n \models p$. It is a finite task to verify whether $\mathfrak{C}_n \in \Omega$, and $\Gamma_n \models p$ or not. Thus, to show that a formula is not a theorem of L we simply proceed (in order) to examine each model until we find a falsifying model.

Combining the positive and negative tests for theoremhood we have the following result. (PDL Pcr and $P?cr$ are excluded because they fail to have the finite Ω -model property.)

3.5.1 Theorem: The problem of telling whether a given formula is a theorem of $P, P?, Pc, Pr, P?c, P?r$ is a decidable problem.

The problem of deciding whether a particular formula is Ω -valid is equivalent to deciding whether its negation is not Ω -satisfiable. So the validity problem is solved if we have a way of deciding satisfiability. We already have given a negative test for validity: the negative test for theoremhood. If in addition, to the finite model property, we also have knowledge of the maximum size of the finite falsifying Ω -model - as the collapsing Ω -model property provides - then the negative test of theoremhood also gives a positive test for Ω -validity. It is clear that quotient models are of size singly exponential in the cardinality of the factoring set. Thus, if the enumeration is in order of increasing size then the algorithm used to examine each finite Ω -model can stop after examining the last model of size $2^{\text{card}(\text{FLC}(\Gamma))}$. (And, moreover, if the logic L is complete and sound with respect to Ω , then we also have a complete method of deciding theoremhood.)

There are two problems with the above algorithm (neither of which gainsay its validity). The first problem is that the complexity of the solution is specified in terms

of the size of $FLC((p))$, not of the length of input p . Fortunately, this problem is eliminated by the Fischer-Ladner lemma (3.4.2). The second problem concerns the complexity of the algorithm. Let p be the formula in question. The number of finite models of size $2^{\text{len}(p)}$ or less is the order of 2 to the power $4^{\text{len}(p)}$ ([PRA80b; pp.263]) - leading to a worst-case deterministic solution of double exponential time complexity. (NOTE: Fischer-Ladner upper bound) Consequently, this algorithm is horrifically impractical (even for "short" p).

In order to examine the complexity of algorithms we adopt the following nomenclature. Let g and f be functions from the natural numbers into the set of real numbers. $g(n)$ is said to be of order $f(n)$, written $O(f(n))$ if there exists a constant c such that $g(n) \leq cf(n)$ for all but some finite number of values for n . Thus, for example, it is well known that the cost (in deterministic time) of checking whether a well-formed formula of length n of the classical propositional logic, CPL, is a theorem is $O(2^n)$.

Let M be one of P , $P?$, Pc and $P?c$ - one of the PDL which has the collapsed Ω -model property.

3.5.2 The Fischer-Ladner upper bound theorem: The Ω -validity problem of a formula p for the logic M is decidable with deterministic time complexity $O(2^{s(p)})$ where $s(p) = c^{\text{len}(p)}$ for some constant $c > 1$.

Proof: Since these PDL satisfy the collapsed Ω -model property. ■

Fischer and Ladner have also determined a lower bound on the complexity of the validity problem for Ω . By building a Σ_{PDL} formula that simulates the behaviour of a *linear-space-bounded alternating Turing machine* (NOTE: Chandra, Stockmeyer & Kozen) they show that the time required to decide whether a formula (of any PDL) is Ω -valid is at least exponential.

3.5.3 The Fischer-Ladner lower bound theorem: Deciding the validity of a formula p for L where $\text{len}(p) = n$ is of deterministic time complexity of at least $O(c^n)$ for some

constant $c > 1$.

Proof: See [F&L79;pp.203-208] for complete details. Also [HAR84;pp.518-519] gives a good synopsis of the proof. ■

The work of Pratt has substantially improved the Fischer-Ladner upper bound for deciding the validity of a formula of PDL M . Pratt's work has culminated in a simple algorithm that, given a formula p of M , builds a finite model (W, π, ν) , for which p is Ω -valid if and only if $p \in w$ for all $w \in W$. This model can be built and searched in deterministic exponential time thus achieving the Fischer-Ladner lower bound to within a polynomial. (NOTE: Pratt's work)

3.5.4 Pratt's upper bound theorem: The Ω -validity problem of a formula p for the logic M is decidable with deterministic time complexity $O(d^{\text{len}(p)})$ for some constant $d > 1$. **Proof:** See [PRA79a; pp.120] but preferably [HAR84;pp.510-517] for details on how to build this model. The construction is similar to our model construction in chapter four. (Our construction, however, retains some "fat".) (NOTE: Harel's upper bound proof) ■

We now report on the decidability of PDL that support *repeat*. (We have already seen that, by virtue of their respective finite Ω -model properties, that Pr and $P?r$ are decidable. However, nothing has been said about the complexity of $Pr/P?r$ validity problem.) The following basic algorithm, due to Streett (NOTE: Streett's work) lends itself to an analysis of complexity. Let R be one of the PDL Pr , $P?r$, Pcr or $P?cr$ and let p be a formula of R . Now R has the so-called tree Ω -model property - which says that Ω models for p can be viewed as labelled graphs which can be transformed ("unraveled" or "unwound") into infinite tree structures of bounded-branching. (This is shown in detail in [STR82;pp.133-137]). Decidability (of p validity) is obtained by constructing an infinite *tree* automaton, say A_p , that accepts all and only all those tree structures of p guaranteed by the tree Ω -model property. (NOTE: tree automaton) This follows directly from the fact that p is Ω -satisfiable if and only if A_p accepts some tree. For Ω -satisfiable p , A_p exists ([STR82;pp.137]) and consequently, the validity problem for R is decidable.

The complexity of the above algorithm depends on (A) the construction of A_p and (B) the determination of whether A_p accepts some tree. The best measure of the cost of (B) is obtained by reducing it to (C) the emptiness problem for infinite tree automaton - which has a known solution of deterministic time complexity $O(c^s)$ where s is the number of states of the automaton in question and $c > 1$ is some constant. (NOTE: Rabin) When R is converse-free (i.e., R is either Pr or P?r) A_p is already an infinite tree automaton and so item (C) is not an issue. In [STR81] and [STR82] the construction of A_p requires triple exponential deterministic time.)

3.5.5 Streett's first upper bound theorem: The Ω -validity problem of a formula p for either Pr or P?r is decidable in triple exponential deterministic time.

Vardi and Stockmeyer have greatly reduced this upper bound to a nondeterministic time complexity $O(c^n)$ where $n = (\text{len}(p))^2$ for some constant $c > 1$. (NOTE: Vardi & Stockmeyer) (This substantial gain supposedly due to an improved method of construction and a simplification of A_p , since they also reduce the problem to one of emptiness for A_p .)

The situation when R is one of Pcr or P?cr is much more complex. It is not clear how two directional computations (i.e., forward and backward tracking) can be checked by infinite tree automaton which, by their nature, can only process their input in one direction. For these cases, Streett invented infinite two-way tree automaton [STR82; pp129-133 ff] that are able to scan their input in two directions. The reduction of two-way A_p to a one-way infinite automaton (required for step (C)) causes a quadruply exponential cost blow-up - i.e., a fourfold exponential increase in the number of states.

3.5.6 Streett's second upper bound theorem: The Ω -validity problem of a formula p for either Pcr or P?cr is decidable in deterministic time at worst octuply exponential in $\text{len}(p)$.

Fortunately, recent work by Vardi has been able to remove the requirement for

Streett's two-way infinite automaton, leading to a solution with non-deterministic time complexity of $O(c^m)$ where $m = (\text{len}(p))^4$. (NOTE: Vardi's work)

Chapter 3 Notes*well-founded relations*

In modal logic (see [BOO79] in particular), an accessibility relation φ is said to be well-founded if it has no infinitely decending φ -sequences i.e., there is no infinite sequence $w_0, w_1, w_2, \dots$ etc. such that $(w_1, w_0) \in \varphi, (w_2, w_1) \in \varphi, \dots, (w_{i+1}, w_i) \in \varphi, \dots$ etc. The converse of a well-founded relation is called well-capped. Therefore, $\nu(\text{repeat}(a))$ can be characterized as the non-well-capped portion of $\pi(a)$.

semantics of loop

In the introduction we discussed the *loop* operator: $\text{loop}(a)$ indicates that program a can enter a non-terminating computation. The semantics for this useful construct is defined inductively as follows: $\nu(\text{loop}(A)) = \emptyset$ for $A \in \Pi_0$; $\nu(\text{loop}(\lambda)) = \emptyset$; $\nu(\text{loop}(a+b)) = \nu(\text{loop}(a)) \cup \nu(\text{loop}(b))$; $\nu(\text{loop}(ab)) = \nu(\text{loop}(a)) \cup \nu(\langle a \rangle \text{loop}(b))$; $\nu(\text{loop}(a^*)) = \nu(\langle a^* \rangle \text{loop}(a)) \cup \nu(\text{repeat}(a))$; $\nu(\text{loop}(p?)) = \emptyset$; $\nu(\text{loop}(A^c)) = \emptyset$ for $A \in \Pi_0$. Thus a program of the form a^* can enter into a non-halting computation during some eventually executed a or because a can itself be repeatedly executed indefinitely. (Note here that, since the semantics of $\text{loop}(a)$ is defined in terms of the semantics of $\text{repeat}(a)$ that *loop* does not add to the expressive power of any PDL containing *Pr*.)

induction model history

Induction models were first introduced by Parikh [PAR78;pp.407] but are embedded deep within his completion proof and are not named. Berman [BER79], on the other hand, has distinguished these models and given them special status within her syntactically motivated *D-axiomatizable* structures. Induction models are called *Loop invariant models* in the subsequent [BER80] and [BER82].

3.1.2 proof

This proof is from [BER82;pp.288-290] which, in turn, is motivated by a result of [PAR78]. Parikh's result implies that for an induction model to be non-standard it must (a) not contain states that cannot be distinguished by some PDL formula and (b) have an infinite number of these non-duplicate states. (Also [BER77;pp.163].)

3.2.13 proof

This example is from [BER79;pp.163].

3.3.1 proof

The models N_n are taken from [SEG78;pp.39] where they are used in a different context (see chapter four, theorem 4.1.14). We also make use of these models to show that logics that support both c and *repeat* operators cannot have the finite-model property (theorem.3.4.8). In [BER78;pp.39] Γ is used as a counter example

to SKM-compactness but this is not demonstrated. See also [HAR84;pp.519] for a related example.

proof of compactness

The compactness of KM, INM and other classes of PDL models are shown to be locally compact in F. Berman's *Syntactic and Semantic Structure in Propositional Dynamic Logic*, Ph.D Dissertation, Seattle: University of Washington (1979) and is sketched (very "sparsely") in [BER79;pp.163-164]. The method is adapted from [PAR78]. The idea is to build a tree of so-called pseudomodels which are themselves trees of finite subsets of Γ which satisfy the conditions of König's lemma - which is the counterpart of compactness for trees that have only finitely many branches leading out of each vertex.

complexity of the consequence problem

Both versions of the consequence problem are not even in the arithmetical hierarchy - whereas global compactness implies that they be recursively enumerable. "This provides a dramatic illustration of the familiar fact that PDL is not compact." [MSM81;pp.3]

chapter 4

The technique of Fischer-Ladner filtration is essential, in one form or another, to demonstrate the completeness of some PDL with respect to the set of standard Kripke models, SKM. (It produces well behaved models - models where all non-standard behaviour has been eliminated.) For those PDL whose completeness can be established by this convenient method, decidability is usually a simple corollary.

FLC extension

Rule g) is from [SEG78;pp.44] and rule h) is from [STR81;pp.380] and [STR82;pp.134].

proof of Fischer-Ladner lemma

The proof given here is an extension of the one given by [F&L77;pp.288-289 (lemma 3.2)] and [F&L79;pp.200-201] where it is shown for $\Gamma \subseteq \Sigma_{PDL}$. The proof has been modified considerably to deal with the *repeat* operator.

Pratt's example

This example is due to Pratt. It is reported in [H&S82;pp.179] and [PRA80b;pp.269]. It is also used in [STR82;pp.126] but is not credited to Pratt.

Rabin

M.O. Rabin, *Decidability of Second Order Theories and Automaton on Infinite Trees*, Transactions of the American Mathematical Society, Vol.141, pp.1-35.

finite models for repeat

For details see [STR81] or [STR82]. This sketch is extracted from the lengthy proof of Pcr decidability (see also, the last part of Section 3.5).

Streett's example

This example (without the satisfying model specified) is from [STR82;pp.126-127].

Fischer-Ladner upper bound

Fischer and Ladner [F&L79] prove that the validity problem has a nondeterministic solution of single exponential time complexity - in fact, using the terminology introduced below, they show an upper bound for this problem of nondeterministic time complexity $O(2^{\text{len}(p)})$. But given the current knowledge of nondeterministic Turing machines the best deterministic solution derivable from their result is no better than the double exponential result we have given above.

Chandra, Stockmeyer & Kozen

Invented by Chandra and Stockmeyer (*Alternation* pp.98-108) and Kozen (*On Parallelism in Turing Machines* pp.89-97). Both of these articles appeared in 17th IEEE Symposium on Foundations of Computer Science (1976).

Pratt's work

Each of [PRA78], [PRA78b] and [PRA79a] arrive at the same complexity result but progressively simplify the algorithm and remove minor inefficiencies. "The method of [PRA78b] is more complicated than is needed just to establish a deterministic one-exponential upper bound. The complications are due to the author's preoccupation with the use of 'bottom-up' methods to achieve good performance on typical cases. (Originally I believed that the worst-case two-exponential upper-bound of Fischer and Ladner would not be substantially improved on, and focused my energies on 'analyzing the structure of the search space' with an eye to improving the typical behaviour of the algorithm, whose *best* case as described by Fischer and Ladner was as bad as its worst. The one-exponential algorithm of [PRA78b] is derived from the 'practical' algorithm in [PRA78], and inherits all of its complications.)" [PRA79a;pp.120 (references changed to our notation)]

Harel's upper bound proof

The proof of [HAR84] is interesting since it neatly combines the algorithm of [PRA79a] with the elementary completeness proof of [K&P81] (see chapter four, section 4.4) to derive this result plus completeness of P? with respect to SKM.

Streett's work

See [STR81] and [STR82] for complete details. Streett invents a new type of automaton for dealing with the analysis of two-way computations (imposed by the converse operator).

tree automaton

Defined in [STR82;pp.127-128 ff] and invented by M.O. Rabin - see (NOTE: Rabin) for source.

Vardi & Stockmeyer

M.Y. Vardi and L. Stockmeyer, *Improved Upper and Lower Bounds for Modal*

Logics of Programs, Proc. of 17th ACM Symposium on Theory of Computing, Providence, May 1985.

Vardi's work

"We show that two-way computations can be viewed as one-way computations with cycles. Thus assertions about two-way computations can be checked by one-way automata." [VAR85;pp.414]

CHAPTER FOUR

COMPLETENESS

In this section we present a method (NOTE: canonical history) for the construction of a model for any consistent normal propositional logic in which all and only its theorems are true. For this reason the so constructed models are called *canonical models*. They also provide the cornerstone for the completeness proof of all the PDLs under our consideration with respect to all Kripke models. But since canonical models are sometimes non-standard, completeness with respect to SKM is not immediate. One method of removing the non-standard aspect of canonical models is known to modal logicians as *filtration*. This method is used successfully in section 4.3 to prove the completeness of PDLs P , $P?$, P_c and $P?c$ with respect to SKM. Unfortunately filtration breaks down for PDLs that incorporate the *repeat* operator and their completeness with respect to SKM remains unknown.

Our canonical completeness proof (sections 4.1, 4.2 and 4.3) follows the approach of [SEG78], which in turn is based on well established and powerful methods used in classical modal logic. (NOTE: canonical completeness proofs) [SEG78] proves the completeness of the logic P (and has an addendum for the completeness of P_c) without the identity program λ . Our proof differs from [SEG78] in respect to the set of equivalent axioms and our choice of $\diamond$ instead of $\Box$ as primitive. We extend Segerberg's work to incorporate the identity program, tests and infinite execution. The work dealing with the *repeat* operator is new and requires an extension to the usual definition of canonical model.

In section 4.4 we consider the completeness proof of Kozen and Parikh [K&P81] and we show how their model constructions are related to our own.

4.1 Canonical Kripke Models

Let N be a normal propositional logic. The canonical model of N is the structure $\mathfrak{C}_N = (U_N, \pi_N, \nu_N)$ where

- (a) $U_N = \{\Delta \mid \text{Max}_N \Delta\}$,
- (b) for $a \in \Pi$, $u, v \in U_N$, $(u, v) \in \pi_N(a)$ iff for all $p \in \Sigma$, $\{ \langle a \rangle p \mid p \in v \} \subseteq u$,
- (c) for all $P \in \Sigma_0$, $u \in U_N$, $u \in \nu_N(P)$ iff $P \in u$,
- (d) if $\Sigma_{\text{PDL, repeat}} \subseteq \Sigma$ then, for all $a \in \Pi$, $u \in U_N$, $u \in \nu_N(\text{repeat}(a))$ iff $\text{repeat}(a) \in u$.

(NOTE: extension of canonical models) Clause (b) may be exchanged with the equivalent

- (b') for $a \in \Pi$, $u, v \in U_N$, $(u, v) \in \pi_N(a)$ iff for all $p \in \Sigma$, $\{p \mid [a]p \in u\} \subseteq v$.

The inclusion of clause (d) means that we are treating propositions of the form $\text{repeat}(a)$ in the same way as primitive $P \in \Sigma$ despite the fact that $a \in \Pi$ may be compound. (We assume that ν_N is automatically extended in the usual way.)

For the remainder of this section let N be a normal propositional logic and let L be a PDL. Set $\mathfrak{C}_N = (U_N, \pi_N, \nu_N)$ and $\mathfrak{C}_L = (U_L, \pi_L, \nu_L)$, the canonical models for logics N and L respectively. We will call upon the properties of maximal extensions (theorem 2.6.1) freely without reference in the proofs that follow.

4.1.1 The Fundamental theorem: For all $u \in U_N$ and all $p \in \Sigma$,

$$\mathfrak{C}_N \models_u p \text{ if and only if } p \in u.$$

Proof: The proof is based upon induction on the complexity of p :

Case 1- $p = P \in \Sigma_0$ or $p = \text{repeat}(a)$: from the definition of $\nu_N(P)$ (clause (c) or (d) respectively) we have $p \in u$ iff $u \in \nu_N(p)$ iff $\mathfrak{C}_N \models_u p$.

Case 2- $p = \neg q$: $\mathfrak{C}_N \models_u \neg q$ iff not $\mathfrak{C}_N \models_u q$ (by 3.1.1b) iff $q \notin u$ (by the induction hypothesis) iff $\neg q \in u$ (since $\text{Max}_N u$).

Case 3- $p = q \vee r$: $\mathfrak{C}_N \models_u q \vee r$ iff $\mathfrak{C}_N \models_u q$ or $\mathfrak{C}_N \models_u r$ (by 3.1.1c) iff $q \in u$ or $r \in u$ (by the induction hypothesis) iff $q \vee r \in u$ (since $\text{Max}_N u$).

Case 4- $p = \langle a \rangle q$: $\mathfrak{C}_N \models_u \langle a \rangle q$ iff $\exists v \in U_N$ such that $(u, v) \in \pi_N(a)$ and $\mathfrak{C}_N \models_v q$ (by 3.1.1d) iff $\exists v \in U_N$ such that $(u, v) \in \pi_N(a)$ and $q \in v$ (by the induction hypothesis) iff $\langle a \rangle q \in u$

(by definition, clause (b)). ■

4.1.2 Corollary: $\vdash_N p$ iff $\mathfrak{C}_N \models p$.

Proof: If $\vdash_N p$ then $\forall w \in U_N, p \in w$ (i.e., $\mathfrak{C}_N \models p$ vis a vis the Fundamental theorem). Conversely, suppose that not $\vdash_N p$ then $\{\neg p\}$ is an N -consistent set. By Lindenbaum's lemma (2.6.2), $\exists w \in U_N$ with $\neg p \in w$ (i.e., $\mathfrak{C}_N \models_w \neg p$ vis a vis the Fundamental theorem). Thus, it is impossible for $\mathfrak{C}_N \models p$. ■

Therefore, if L is one of the propositional dynamic logics we have defined so far, then $\mathfrak{C}_L$, the canonical model of the logic L , certainly exists. And, according to the Fundamental theorem, if p is not a theorem of L then not $\mathfrak{C}_L \models p$ i.e., $\mathfrak{C}_L$ is an invalidating model for the logic L . This fact gives rise to the following general completeness result:

4.1.3 Corollary: If $\mathfrak{C}_L \in \Omega$ then the propositional dynamic logic L is complete with respect to Ω i.e., if q is Ω -valid then $\vdash_L q$.

We demonstrate below that for any propositional dynamic logic L , $\mathfrak{C}_L$ is a Kripke model (theorems 4.1.4 through to 4.1.12 inclusive), and consequently L is complete with respect to KM , the class of all Kripke models. As it happens $\mathfrak{C}_P$, the canonical model for the basic PDL P , is a non-standard induction model, and so we have the fine grained results that P , and ultimately all its extensions, are complete with respect to SKM and INM .

4.1.4 Theorem: For all $a, b \in \Pi$, $\pi_L(a+b) = \pi_L(a) \cup \pi_L(b)$ (i.e., π_L satisfies the union condition).

Proof: $(u, v) \in \pi_L(a) \cup \pi_L(b)$ iff if $p \in v$ then $\langle a \rangle p \in u$ or $\langle b \rangle p \in u$ iff if $p \in v$ then $\langle a \rangle p \vee \langle b \rangle p \in u$ (since $\text{Max}_L v$) iff if $p \in v$ then $\langle a+b \rangle p \in u$ (by $Ax. +$) iff $(u, v) \in \pi_L(a+b)$. ■

4.1.5 Theorem: For all $a, b \in \Pi$, $\pi_L(ab) = \pi_L(a) \circ \pi_L(b)$ (i.e., π_L satisfies the composition condition).

Proof: If $(u, v) \in \pi_L(a) \circ \pi_L(b)$ then $\exists z \in U_L$ with $(u, z) \in \pi_L(a)$ and $(z, v) \in \pi_L(b)$. Therefore, if

$p \in v$ then $\langle a \rangle \langle b \rangle p \in u$, and, since $Ax.$; and Max_L , then $\langle ab \rangle p \in u$. Consequently $(u, v) \in \pi_L(ab)$.

Conversely, let

$$(1) \quad (u, v) \in \pi_L(ab).$$

Assume some fixed and complete enumeration of all the formulae of v , say

$$r_1, r_2, r_3, \dots \text{ etc.}$$

and, inductively, define another sequence of formulae as follows:

$$\begin{aligned} q_1 &= r_1, \\ q_{n+1} &= q_n \& r_{n+1}. \end{aligned}$$

Clearly, since $Max_L v$, we have

$$(2) \quad \text{for all } n < \omega, q_n \in v$$

Also, by PC, for all n and i , $\vdash_L q_{n+i} \rightarrow q_n$. Thus, by RB, $\vdash_L \langle b \rangle q_{n+i} \rightarrow \langle b \rangle q_n$. By the deduction theorem 2.3.1e this means that

$$(3) \quad \text{for all } n \text{ and } i, (\langle b \rangle q_{n+i}) \vdash_L \langle b \rangle q_n.$$

Now consider the sets

$$\begin{aligned} \alpha &= \{p \mid [a]p \in u\}, \\ \beta &= \{\langle b \rangle q_n \mid n < \omega\}, \\ \Delta &= \alpha \cup \beta. \end{aligned}$$

and suppose that Δ is L-inconsistent, i.e., there is a finite subset of Δ , say Δ' , which is also L-inconsistent (by 2.3.2). Let

$$\begin{aligned} \alpha' &= \{p_1, p_2, \dots, p_m\} = \alpha \cap \Delta' \\ \beta' &= \Delta' - \alpha' \end{aligned}$$

and let $\langle b \rangle q$ be the formula of β' with the largest index. According to (3) then $\vdash_L p_1 \& \dots \& p_m \& \langle b \rangle q \rightarrow \perp$, or by PC, $\vdash_L p_1 \& \dots \& p_m \rightarrow \neg \langle b \rangle q$. By applying RM and 2.4.3a (extended to cover conjunctions of finite length), we have $\vdash_L [a]p_1 \& \dots \& [a]p_m \rightarrow \neg \langle a \rangle \langle b \rangle q$. Since each of $[a]p_1, \dots, [a]p_m \in u$ and because $Max_L u$, then $\neg \langle a \rangle \langle b \rangle q \in u$. Therefore, since $Max_L u$, $\langle a \rangle \langle b \rangle q \notin u$ and so, by $Ax.$, $\langle ab \rangle q \notin u$. However, by (1), if $q \in v$ then $\langle ab \rangle q \in u$ therefore $q \notin v$ - which directly contradicts (2). Consequently Δ must be L-consistent. Since Δ is L-consistent then, by Lindenbaum's lemma, there is some $z \in U_L$ with $\Delta \subseteq z$. Since $\Delta \subseteq z$, then according to theorem 2.6.4, $\{\langle a \rangle p \mid p \in z\} \subseteq u$. Therefore, $(u, z) \in \pi_L(a)$. Also, for any $r_i \in v$, $\vdash_L \langle b \rangle q_i \rightarrow \langle b \rangle r_i$ (by PC and RB), therefore, since $\beta \subseteq z$, $\{\langle b \rangle p \mid p \in v\} \subseteq z$, so that $(z, v) \in \pi_L(b)$. Therefore,

by definition, $(u,v) \in \pi_L(a) \circ \pi_L(b)$ as required. ■

4.1.6 Theorem: For all $a \in \Pi$, $\forall u \in U_N$, $(u,u) \in \pi_L(a^*)$ (i.e., π_L satisfies the $*$ -reflexive condition).

Proof: Assume that $p \in u$ where $u \in U_L$. Then, by 2.5.1b, MP and since $\text{Max}_L u$, $\langle a^* \rangle p \in u$. Therefore, by definition $(u,u) \in \pi_L(a^*)$. ■

4.1.7 Theorem: For all $a \in \Pi$, $\pi_L(a^*) \circ \pi_L(a^*) \subseteq \pi_L(a^*)$ (i.e., π_L satisfies the $*$ -transitive condition).

Proof: Let $(u,w) \in \pi_L(a^*) \circ \pi_L(a^*)$ (i.e., there is a $v \in U_L$ such that $(u,v) \in \pi_L(a^*)$ and $(v,w) \in \pi_L(a^*)$, i.e., $\langle a^* \rangle p \mid p \in v \subseteq u$ and $\langle a^* \rangle p \mid p \in w \subseteq v$ respectively). We are required to show that $(u,w) \in \pi_L(a^*)$, i.e., $\langle a^* \rangle p \mid p \in w \subseteq u$. Let $q \in w$, then $\langle a^* \rangle q \in v$ and moreover $\langle a^* \rangle \langle a^* \rangle q \in u$. Since $\text{Max}_L u$ we have, by 2.5.4b and MP, that $\langle a^* \rangle q \in u$. ■

4.1.8 Theorem: For all $a \in \Pi$, $\pi_L(a) \subseteq \pi_L(a^*)$ (i.e., π_L satisfies the $*$ -inherit condition).

Proof: Let $(u,v) \in \pi_L(a)$. Thus, if $p \in v$ then $\langle a \rangle p \in u$ and since $\text{Max}_L u$ we have, by 2.5.3b and MP, that $\langle a^* \rangle p \in u$. Thus $\langle a^* \rangle p \mid p \in v \subseteq u$. ■

4.1.9 Theorem: $\pi_L(\lambda) = \{(w,w) \mid w \in W\}$ (i.e., π_L satisfies the λ -identity condition).

Proof: Let $x \in U_L$. By Ax.id and $\text{Max}_L x$ we have $\langle \lambda \rangle p \in x$ iff $p \in x$. ■

4.1.10 Theorem: For all $p \in \Sigma_{PDL,?} \subseteq \Sigma$, $\pi_L(p?) = \{(w,w) \mid w \in \nu_L(p)\}$ (i.e., π_L satisfies the test condition).

Proof: Suppose that $w \in U_L$, $p \in w$ and $q \in w$. Since $\text{Max}_L w$ then, by Ax.?, $\langle p? \rangle q \in w$. Therefore $(w,w) \in \pi_L(p?)$. ■

4.1.11 Theorem: For all $a \in \Pi_{PDL,c} \subseteq \Pi$, $\pi_L(a^c) = \pi_L^{-1}(a)$ (i.e., π_L satisfies the inverse condition).

Proof: First, we assume that $(u,v) \in \pi_L(p^c)$. Now suppose that $p \in u$, then, by Ax.c and $\text{Max}_L u$, $[a^c] \langle a \rangle p \in u$, i.e., $\langle a^c \rangle \neg \langle a \rangle p \in u$ (again by $\text{Max}_L u$). By definition, this implies that $\neg \langle a \rangle p \notin v$, and because $\text{Max}_L v$, $\langle a \rangle p \in v$. Therefore, $(v,u) \in \pi_L(a)$ or $(u,v) \in \pi_L(a)^{-1}$. Conversely, suppose that $(u,v) \in \pi_L(a)^{-1}$, i.e., $(v,u) \in \pi_L(a)$, and also assume that $p \in u$. By

Ax.c and $\text{Max}_L u$, we have that $[a] \langle a^c \rangle p \in u$, i.e., $\langle a \rangle \neg \langle a^c \rangle p \notin u$. Therefore, $\neg \langle a \rangle p \notin v$, by definition, and since $\text{Max}_L v$, $\langle a^c \rangle p \in v$. Consequently $(u, v) \in_L (a^c)$. ■

4.1.12 Theorem: For all $a \in \Pi_{\text{PDL}}$, $\pi_L(a^*) \supseteq \bigcup_{n < \omega} (\pi_L(a)^n)$.

Proof: By induction on n .

For $n=0$, the result is immediate from 4.1.6 since $\pi_L(a)^0$ is just $\pi_L(\lambda)$ (4.1.9).

Now suppose that $\pi_L(a)^n \subseteq \pi_L(a^*)$. If $(x, z) \in \pi_L(a)^{n+1} = \pi_L(a) \circ \pi_L(a)^n$ then there is a $y \in U_L$ with $(x, y) \in \pi_L(a)$ and $(y, z) \in \pi_L(a)^n$. But since $(x, y) \in \pi_L(a^*)$ by 4.1.8, and $(y, z) \in \pi_L(a^*)$ by the induction hypothesis then also $(x, z) \in \pi_L(a^*)$ by 4.1.7. ■

Unfortunately the converse of Theorem 4.1.12 does not hold and consequently we are not guaranteed that $\mathfrak{C}_L$ is a standard Kripke model.

4.1.13 Theorem: There are non-standard canonical Kripke models. Indeed $\mathfrak{C}_P$, the canonical model of P , is non-standard.

Proof: Let $P \in \Sigma_0$ and $a \in \Pi_{\text{PDL}}$. Let $\mathfrak{C}_P = (U_P, \pi_P, \nu_P)$ be the canonical model for the logic P . Let $N_n = (N, R, \nu_n)$ be the standard Kripke model where

- a) N is the set of natural numbers,
- b) $(n, m) \in R(a)$ iff $m = n+1$ (i.e., $R(a)$ is the relation of immediate successor),
- c) $\nu_n(P) = \{i \mid i \leq n\}$ (i.e., valuation depends only upon $n \in N$).

Now consider the set $\Gamma = \{[a]^n P \mid n < \omega\} \cup \{[a]^* P\}$. For any non-empty finite subset Γ' of Γ there is a $j \in N$ such that $[a]^j P \in \Gamma'$ but for any $k \in N$, $[a]^{j+k+1} P \notin \Gamma'$. Therefore Γ' is (globally) SKM-satisfied and consequently Γ' is P -consistent. Indeed, since Γ' is arbitrary then all finite subsets of Γ are P -consistent, implying that Γ itself is P -consistent (2.3.2). Thus we can extend Γ to some $x \in U_P$. Now since $\neg[a]^* P \in x$ (i.e., $\langle a^* \rangle \neg P \in x$), there is a $y \in U_P$ such that $(x, y) \in \pi_P(a^*)$ but $P \notin y$ (by the definition of π_P). However, if $(x, y) \in \pi_P(a)^n$ for some $n < \omega$ then $\langle a \rangle^n p \mid p \in y \subseteq x$ or $\{p \mid [a]^n p \in x\} \subseteq y$ (by 2.6.4). But since $P \notin y$ then $[a]^n P \notin x$ which contradicts the fact that $\Gamma \subseteq x$. Therefore, $(x, y) \notin \pi_P(a)^n$ for any $n < \omega$. (NOTE: Segerberg's example) ■

Therefore canonical models are not always standard. In the next section we

introduce a procedure which eliminates any "non-standard" behaviour from a canonical model for a given set of *repeat-free* formulae.

Despite this negative result, we have shown that **KM** and, indeed, **non-SKM** are non-empty (i.e., there are Kripke models, however non-standard). This, in turn, demonstrates that **P**, **P?**, **Pc**, **Pr**, **P?c**, **P?r**, **Pcr** and **P?cr** are all consistent (3.2.14). Furthermore, the existence of **DKM** models implies the consistency of the logics **D**, **D?**, **Dc**, **Dr**, **D?c**, **D?r**, **Dcr** and **D?cr**, through 3.2.17. This is the point of the next theorem.

4.1.14 Theorem: Let $\Gamma_D = (U_D, \pi_D, \nu_D)$ be the canonical model of the logic **D**, then π_D satisfies the serial condition.

Proof: Let $v \in W$ and $p \in v$. Since $\text{Max}_D v$, by **RP** we have $\langle a \rangle p \in v$ for all $a \in \Pi_{pDL}$. Thus, by 3.1.1d, there is a $w \in W$ such that $(v, w) \in \pi_D(a)$ and $p \in w$. The former of these is exactly what we are required to prove. ■

Finally we are also in a position to assert that **P** and all its extensions so far defined are complete with respect to the set of models **KM**, **non-SKM** and **INM**.

4.1.15 The KM (non-SKM and INM) completeness theorem: Let **L** be one of **P**, **P?**, **Pc**, **Pr**, **P?c**, **P?r**, **Pcr** or **P?cr**. If q is **KM** (**non-SKM** or **INM**)-valid then $\vdash_L q$.

Proof: By 4.1.3, directly from the fact that $\mathfrak{C}_L$ is a (non-standard) Kripke model. ■

4.2 Fischer-Ladner Filtrations

In the last section we showed that the canonical models of **P** and its extensions are not standard - and, consequently, we are short of our goal of proving completeness with respect to standard Kripke models. In this section, we modify a technique from modal logic which will (sometimes) enable us to remove non-standard behaviour from canonical models but retain its "essential truths". The demonstration of this is

undertaken in the section following this. Unfortunately, our method produces the sort of finite models that prevent the retention of essential truths expressed by the *repeat* operator. The reasons for this we have already examined in chapter three (section 3.4). There we found that the logics Pr , $P?r$, Pcr and $P?cr$ lacked the collapsed SKM-model property because finite quotient models of certain infinite models of $\neg repeat(a)$ did not preserve its truth value (theorem 3.4.6). (NOTE: completeness of PDLs with *loop*)

For the duration of this section let M be any of the logics P , $P?$, Pc and $P?c$ (i.e., M does not support the *repeat* operator) and let F be the Fischer-Ladner closure of a set of formulae Γ (i.e., $F = FLC(\Gamma)$). Thus $\mathfrak{C}_M = (U_M, \pi_M, \nu_M)$ is the canonical model for M .

A Fischer-Ladner filtration through F of $\mathfrak{C}_M$, denoted $\mathfrak{C}_M^F = (U_M^F, \pi_M^F, \nu_M^F)$, is a model that satisfies the following conditions:

- (a) $U_M^F = U_M/F$, i.e., U_M^F is the partition of U_M induced by $\simeq$ (U_M^F is the set of all equivalence class modulo $\simeq_F$).
- (b) for all $a \in \Pi$ such that $\langle a \rangle q \in F$, if $(u, v) \in \pi_M(a)$ then $(|u|, |v|) \in \pi_M^F(a)$,
- (c) for all $P \in \Sigma_0 \cap F$, $\nu_M^F(P) = \{|u| \mid u \in \nu_M(P)\}$.
- (d) for all $a \in \Pi$ such that $\langle a \rangle q \in F$, if $(|u|, |v|) \in \pi_M^F(a)$ then $\{\langle a \rangle p \mid p \in v\} \cap F \subseteq u$.

Thus $\mathfrak{C}_M^F$ is very much like the F -Quotient model of $\mathfrak{C}_M$ (see chapter 3, section 3.4). In fact the filtration technique generates a family of models of which the F -Quotient model is a special case. As in the case with $\mathfrak{C}_M/F$, note that π_M^F is well defined. Recall, also that if F has n members then U_M^F has at most 2^n members.

Let $\mathfrak{C}_M^F = (U_M^F, \pi_M^F, \nu_M^F)$ be any Fischer-Ladner filtration through F of $\mathfrak{C}_M$. For the same reason that M enjoys the collapsed SKM-model property, $u \in U_M$ and $|u| \in U_M^F$ agree upon which formulae of Σ are true and which are not (for $\mathfrak{C}_M$ and $\mathfrak{C}_M^F$ respectively).

4.2.1 The (Fischer-Ladner) Filtration theorem: For all $p \in F$, $\mathfrak{C}_M^F \models_{|u|} p$ iff $\mathfrak{C}_M \models_u p$.

Proof: The proof is similar to that of 3.4.3 ($\mathfrak{C}_M^F$ acts as like an F -Quotient model of

$\mathfrak{C}_M$ up to ν_M^F modulo F). ■

4.2.2 Corollary: For all $p \in F$, $\mathfrak{C}_M^F \models_{[u]} p$ iff $p \in u$.

Proof: By the Fundamental theorem. ■

We now narrow our attention to a class of models which prove to be Fischer-Ladner filtrations through F of $\mathfrak{C}_M$ for which there are no non-standard features. It is the proof that such models are indeed filtration models that determines the completeness of M with respect to SKM.

A finest Fischer-Ladner filtration through F of $\mathfrak{C}_M$ is a model (U_M^F, ξ, ν_M^F) , where ξ satisfies the following condition: For all $A \in \Pi_0$ such that $\langle A \rangle p \in F$ for some $p \in \Sigma$,

$$([u], [v]) \in \xi(A) \text{ iff } \exists u', v' \text{ such that } u' \simeq u, v' \simeq v \text{ and } (u', v') \in \pi_M(A).$$

A Segerberg model of M through F is any standard Kripke model, say $\mathfrak{R} = (W, \xi, \tau)$ that is a finest Fischer-Ladner filtration through F of $\mathfrak{C}_M$. Segerberg models certainly exist by virtue of the construction. Indeed there will usually be many such models since formulae outside of F will be unconstrained by the construction. An example of one particularly important Segerberg model is given in section 4.4.

4.3 Completeness With Respect to Standard Kripke Models

Throughout this section let M be one of P , $P?$, Pc or $P?c$. Then M satisfies the collapsed SKM-model property. Let $\mathfrak{C}_M = (U_M, \pi_M, \nu_M)$ be the canonical model for M , F be the Fischer-Ladner closure of a set of formulae Γ (i.e., $F = \text{FLC}(\Gamma)$) and $\mathfrak{C}_M^F = (U_M^F, \pi_M^F, \nu_M^F)$ be a Fischer-Ladner filtration through F of $\mathfrak{C}_M$.

To prove the completeness of the logic M (with respect to SKM) we show for any non-theorem of M that there is a standard Kripke model in which it is not true. This

model is called an invalidating model of the non-theorem. In our proof, this model turns out to be a Segerberg model of the non-theorem of the canonical model of M .

We demonstrate that, where $F = \text{FLC}(\Gamma)$ is a finite set then any Segerberg model of M through Γ , say $\mathfrak{R} = (W, \xi, \tau)$, is a Fischer-Ladner filtration of $\mathfrak{C}_M$ through F . This result, plus the filtration theorem (4.2.1/2) will give us the completeness result. Since $W = U_M^F$ and $\tau = \nu_M^F$ we are only required to prove that ξ satisfies clauses (b) and (d) of the definition of Fischer-Ladner filtration of $\mathfrak{C}_M$ through F (lemmas 4.3.2 and 4.3.4 respectively). For the remainder of this section let $\mathfrak{R} = (W, \xi, \tau)$ be a Segerberg model of M through Γ .

4.3.1 Lemma: Let $u \in U_M$ and $a \in \Pi$. One can find a formula t such that for any $w \in U_M$, $t \in w$ iff $(|u|, |w|) \in \xi(a)$.

Proof: Let $u \in U_M$ and $a \in \Pi$. Since F is finite then so too must $W = U_M^F$. Define two mutually exclusive subsets of W :

$$X = \{x \in W \mid (|u|, |x|) \in \xi(a)\},$$

$$X' = \{x \in W \mid (|u|, |x|) \notin \xi(a)\}.$$

(Note that $X \cup X' = W$.) Let $X = \{x_1, \dots, x_n\}$ and $X' = \{x_{n+1}, \dots, x_{m+n}\}$, i.e., $\{x_1, \dots, x_n, x_{n+1}, \dots, x_{m+n}\}$ is a partition of U_M induced by $\simeq_F$ and thus, for $1 \leq i, j \leq m$, $x_i = x_j$ iff $i = j$. When not $i = j$ then, for all $w \in x_i$, $w' \in x_j$ there exist a $t_{ij} \in w \cap F$ with $t_{ij} \notin w' \cap F$. Also, set $t_{ii} = \top$, $1 \leq i \leq m$. For any $1 \leq i \leq n$ consider the formula

$$t_i = t_{i1} \& t_{i2} \& \dots \& t_{in}.$$

Now, for $1 \leq i \leq n$, $t_i \in w$ iff $w \in x_i$. Therefore

$$t_1 \vee t_2 \vee \dots \vee t_n \in w \text{ iff } w \in x_1 \text{ or } w \in x_2 \text{ or } \dots \text{ or } w \in x_n.$$

Thus, our required formula is $t = t_1 \vee t_2 \vee \dots \vee t_n$. ■

4.3.2 Lemma: For all $a \in \Sigma$ such that $\langle a \rangle q \in F$, if $(u, v) \in \pi_M(a)$ then $(|u|, |v|) \in \xi(a)$.

Proof: The proof is based upon induction on the complexity of a .

(Notice that in case 6 where $a = p?$ the proof makes no use of the induction hypothesis.)

Case 1- $a = A$, $A \in \Pi_0$: Let $(u, v) \in \pi_M(A)$. Since $u \simeq u$ and $v \simeq v$ then, by definition, $(|u|, |v|) \in \xi(A)$.

Case 2- $a = \lambda$: Let $(u, v) \in \pi_M(\lambda)$. As in case 1, $(|u|, |v|) \in \xi(\lambda)$. Also $u = v$ (by 4.1.9) thus

$|u|=|v|$ as required.

Case 3- $a=b+c$: Let $(u,v) \in \pi_M(b+c)$. Thus $(u,v) \in \pi_M(b)$ or $(u,v) \in \pi_M(c)$ (by 4.1.4) and so, by the induction hypothesis, $(|u|,|v|) \in \xi(b)$ or $(|u|,|v|) \in \xi(c)$. Now, since ξ satisfies the union condition of all Kripke models then $(|u|,|v|) \in \xi(b+c)$.

Case 4- $a=bc$: Let $(u,v) \in \pi_M(bc)$. Thus $(u,v) \in \pi_M(b) \circ \pi_M(c)$ (by 4.1.5), i.e., $\exists z \in U_F$ such that $(u,z) \in \pi_M(b)$ and $(z,v) \in \pi_M(c)$. Thus, by the induction hypothesis, $(|u|,|z|) \in \xi(b)$ and $(|z|,|v|) \in \xi(c)$, i.e., $(|u|,|v|) \in \xi(b) \circ \xi(c)$. Now, since ξ satisfies the composition condition of all Kripke models then $(|u|,|v|) \in \xi(bc)$.

Case 5- $a=b^*$: Let

$$(1) \quad (u,v) \in \pi_M(b^*)$$

but assume, in order to arrive at a contradiction, that

$$(2) \quad (|u|,|v|) \notin \xi(b^*).$$

Let t be the formula insured by lemma 4.3.1. Therefore, from (2), $t \notin v$ and hence, from (1) and 3.1.1e,

$$(3) \quad [b^*]t \notin u.$$

Since ξ satisfies the $*$ -reflexive condition (i.e., $(|u|,|u|) \in \xi(b^*)$) then $t \in u$. Since $\text{Max}_M u$, then by IN, $[b^*](t \rightarrow [b]t) \notin u$, or equivalently,

$$\langle b^* \rangle (t \& \langle b \rangle \neg t) \in u.$$

By the Fundamental theorem and 3.1.1d, there exists $x \in U_M$ such that $(u,x) \in \pi_M(b^*)$ and

$$(4) \quad t \& \langle b \rangle \neg t \in x$$

Then (since $\text{Max}_M x$)

$$(5) \quad t \in x$$

and (from (4) by the Fundamental theorem and 3.1.1d once more) there exists $y \in U_M$ such that

$$(6) \quad (x,y) \in \pi_M(b)$$

$$(7) \quad t \notin y.$$

From (5), by 4.3.1,

$$(8) \quad (|u|,|x|) \in \xi(b^*).$$

From (6), by the induction hypothesis, $(|x|,|y|) \in \xi(b)$. Thus according to the $*$ -inherit condition,

$$(9) \quad (|x|,|y|) \in \xi(b^*).$$

From the fact that ξ satisfies $*$ -transitivity, this result and (8) imply that $(|u|, |v|) \in \xi(b^*)$, or, according to 4.3.1, $t \in y$ which contradicts (7). Consequently $(|u|, |v|) \in \xi(b^*)$.

Case 6- $a=p?$: Let $(u, v) \in \pi_M(p?)$. By 4.1.10, $u=v$ and $u \in \nu_M(p)$. Therefore $|u|=|v|$. Also, by the filtration theorem (4.2.1), $|u| \in \tau(p)$, and consequently, $(|u|, |v|) \in \xi(p?)$.

Case 7- $a=b^c$: Let $(u, v) \in \pi_M(b^c)$. By 4.1.11, $(u, v) \in \pi_M(b)^{-1}$ which means that $(v, u) \in \pi_M(b)$. By the induction hypothesis, $(|v|, |u|) \in \xi(b)$, and so $(|u|, |v|) \in \xi(b)^{-1}$. Since ξ satisfies the converse condition then $(|u|, |v|) \in \xi(b^c)$.

■

4.3.3 Lemma: For all i , if $(|x|, |y|) \in \xi(a)^i$ then, $\langle a^* \rangle p \in y$ implies that $\langle a^* \rangle p \in x$.

Proof: By induction on i . When $i=0$ the result is immediate from the definition of $\xi(a)^0$. Assume the result holds for $i \leq n$. Suppose that $\langle a^* \rangle p \notin x$. Since $\text{Max}_M x$ then, by 2.5.2b,

$$(1) \quad \langle a \rangle \langle a^* \rangle p \notin x.$$

Now, let $(|x|, |y|) \in \xi(a)^{n+1}$, i.e., $(|x|, |y|) \in \xi(a) \circ \xi(a)^n$. Thus, there is a $z \in U_n$ such that

$$(2) \quad (|x|, |z|) \in \xi(a)$$

and

$$(3) \quad (|z|, |y|) \in \xi(a)^n.$$

From (1) and (2), by the Fundamental theorem and 3.1.1d, $\langle a^* \rangle p \notin z$. Thus, from (3) via the induction hypothesis, $\langle a^* \rangle p \notin y$ - as required. ■

4.3.4 Lemma: For all $a \in \Sigma$ such that $\langle a \rangle q \in F$, if $(|u|, |v|) \in \xi(a)$ then $\{\langle a \rangle p \mid p \in v\} \cap F \subseteq u$.

Proof: The proof is based upon induction on the complexity of a .

(Notice that in cases 5 and 6 where $a=b^*$ and $a=p?$ respectively, the proof makes no use of the induction hypothesis.)

Case 1- $a=A$, $A \in \Pi_0$: Let $p \in v$ and $\langle A \rangle p \in F$. If $(|u|, |v|) \in \xi(A)$ then $\exists u', v'$ such that $u' \simeq u$, $v' \simeq v$ and $(u', v') \in \pi_M(A)$ (by definition) i.e., $\{\langle A \rangle q \mid q \in v'\} \subseteq u'$. Therefore $\langle A \rangle p \in u' \cap F = u \cap F$.

Case 2- $a=\lambda$: Let $p \in v$ and $\langle \lambda \rangle p \in F$. If $(|u|, |v|) \in \xi(\lambda)$ then $u \simeq v$ (4.1.9) and $\exists w \in U_M$ such that $w \simeq u$ and $(w, w) \in \pi_M(\lambda)$ (by definition) i.e., $\{\langle \lambda \rangle q \mid q \in w\} \subseteq w$. Therefore $\langle \lambda \rangle p \in F = w \cap F = u \cap F$.

Case 3- $a=b+c$: Let $\langle b+c \rangle p \in F$ and $p \in v$. If $(|u|, |v|) \in \xi(b+c)$ then $(|u|, |v|) \in \xi(b)$ or $(|u|, |v|) \in \xi(c)$

(since ξ satisfies the union condition). Also, by the definition of F , $\langle b \rangle p \in F$ and $\langle c \rangle p \in F$. Thus, by the induction hypothesis, we have $\langle b \rangle p \in u$ or $\langle c \rangle p \in u$, and so $\langle b+c \rangle p \in u$ (since $\text{Max}_M u$ and $\text{Ax.}+$).

Case 4- $a=bc$: Let $(|u|, |v|) \in \xi(bc)$, $\langle bc \rangle p \in F$ and $p \in v$. Since ξ satisfies the composition condition, $\exists |w| \in W$ such that $(|u|, |w|) \in \xi(b)$ and $(|w|, |v|) \in \xi(c)$. Also, by the definition of F , $\langle b \rangle \langle c \rangle p \in F$. Thus by the induction hypothesis, $\langle c \rangle p \in w$ and then $\langle b \rangle \langle c \rangle p \in u$. By the fact $\text{Max}_M u$ and Ax. , $\langle bc \rangle p \in u$.

Case 5- $a=b^*$: Let $(|u|, |v|) \in \xi(b^*)$, $\langle b^* \rangle p \in F$ and $p \in v$. Since $\text{Max}_M v$, by 2.5.1b then $\langle b^* \rangle p \in v$ and so, according to lemma 4.3.3, $\langle b^* \rangle p \in u$.

Case 6- $a=p?$: Let $(|u|, |v|) \in \xi(p?)$ and $\langle p? \rangle q \in F$. Since ξ satisfies the test condition then $|u|=|v|$ and $|u| \in \tau(p)$ (i.e., $|u| \in \nu_M^F(p)$). Therefore, $(|u|, |v|) \in \pi_M^F(p?)$ (by definition). By condition (d) of the definition of $\mathfrak{C}_M^F$ (section 4.2) we know that $\{\langle p? \rangle q | q \in v\} \cap F \subseteq u$, as required.

Case 7- $a=b^c$: Let $(|u|, |v|) \in \xi(b^c)$, $\langle b^c \rangle p \in F$ and $p \in v$. Since ξ satisfies the inverse condition, $(|v|, |u|) \in \xi(b)$ and since $\langle b^c \rangle p \in F$ then, according to the definition of F , $\langle b \rangle [b^c] \neg p \in F$. Suppose that $\langle b^c \rangle p \notin u$ i.e., $[b^c] \neg p \in u$. Thus, by the induction hypothesis, $\langle b \rangle [b^c] \neg p \in v$. By theorem 2.5.10 and $\text{Max}_M v$, $\neg p \in v$ and so $p \notin v$. But this contradicts the initial assumption and therefore $\langle b^c \rangle p \in u$, as required.

4.3.5 The SKM completeness theorem: Let M be one of P , $P?$, Pc or $P?c$. If q is SKM-valid then $\vdash_M q$.

Proof: Assume not $\vdash_M q$. Then, by 4.1.2 there is some $x \in U_M$ such that $q \notin x$. Let $\mathfrak{R} = (W, \xi, \tau)$ be an Segerberg model of M through $\{q\}$. Since $\text{FLC}(\{q\})$ is finite then $\mathfrak{R}$ is a Fischer-Ladner filtration of $\mathfrak{C}_M$ through $\text{FLC}(\{q\})$. Thus, by the Filtration theorem, not $\exists |x| q$, and since $\mathfrak{R} \in \text{SKM}$ this completes the proof. ■

By combining theorems 3.2.11, 3.2.12, 4.1.15 and 4.3.5 we may summarize the results so far.

4.3.6 Determination theorem: Let M be one of P , $P?$, Pc or $P?c$. Let $q \in \Sigma$, q is SKM-valid iff $\vdash_M q$.

q is INM-valid iff $\vdash_M q$.

4.4 The Kozen-Parikh Model

In their paper [K&P81], Kozen and Parikh give an elementary proof of the completeness of the logic P . (NOTE: Harel's proof) In a single succinct step they neatly construct an invalidating model for any non-theorem of P . In this section we show their construction to be a distinguished Segerberg model (modulo isomorphism).

As in the previous section, let M be one of P , $P?$, Pc or $P?c$ for the duration. (That is, M has the collapsed SKM-model property.) Let q be any formula and let F be the Fischer-Ladner closure of $\{q\}$, i.e., $F = FLC(\{q\})$. F is finite so we can consider it to be the indexed set

$$(1) \quad F = (q_1, q_2, \dots, q_n).$$

An Atom of F is a formula, say p , which is a M -consistent (i.e., not $(p) \vdash_M \perp$) boolean conjunction of formulae of the form $p = p_1 \& p_2 \& \dots \& p_n$, where p_i ($1 \leq i \leq n$) is either q_i or $\neg q_i$. Let At_F denote the set of all atoms of F (NOTE: atom).

4.4.1 Lemma: for any $s, t \in At_F$ and $A \in \Pi_0$, if the set $\{s, \langle A \rangle t\}$ is M -consistent then $s = t$.
Proof: Let $s = s_1 \& \dots \& s_n$ and $t = t_1 \& \dots \& t_n$, and suppose, for some $1 \leq k \leq n$, that not $s_k = t_k$. First assume that $s_k = q_k$ and $t_k = \neg q_k$. Now if $\{s, \langle A \rangle t\}$ is M -consistent then, by 2.4.5 (extended to cover any finite conjunction) and PC , it must be that $\{q_k, \langle A \rangle \neg q_k\}$ is also M -consistent. However, by A -necessitation RN , then $\{[A]q_k, \neg [A]q_k\}$ is M -consistent, and this is obviously false. A similar argument shows that if $s_k = \neg q_k$ and $t_k = q_k$ then the set $\{\neg \langle A \rangle q_k, \langle A \rangle q_k\}$ is M -consistent, which again is obviously untrue. Therefore $\{s, \langle A \rangle t\}$ must be M -inconsistent. ■

In [K&P81], Kozen and Parikh define a standard Kripke model $S = (At_F, \gamma, \nu)$ where $F = FLC(\{q\})$ for some $q \in \Sigma$ and, for $s, t \in At_F$, $A \in \Pi_0$ and $P \in \Sigma_0$

$(s, t) \in \gamma(A)$ iff the formula $s \& \langle A \rangle t$ is M -consistent

$$\mathcal{U}(P) = \{ s \in \text{At}_F \mid \{s\} \vdash_M P \},$$

and where γ is extended to cover the case of compound programs in the standard way, i.e., let $a, b \in \Pi$ and $s, t \in \text{At}_F$,

- (a) $(s, t) \in \gamma(a+b)$ iff $(s, t) \in \gamma(a) \cup \gamma(b)$,
- (b) $(s, t) \in \gamma(ab)$ iff $(s, t) \in \gamma(a) \circ \gamma(b)$,
- (c) $(s, t) \in \gamma(a^*)$ iff for some $k \leq \omega$, $(s, t) \in \gamma(a)^k$.

[K&P81] do not consider the special program λ or the **OpProg** operators $?$ and c . However, since their inclusion causes no additional difficulty we extend $\mathfrak{S}$ so it also satisfies the identity, test and converse conditions:

- (d) $(s, t) \in \gamma(\lambda)$ iff $s = t$,
- (e) $(s, t) \in \gamma(p?)$ iff $s = t$ and $s \in \mathcal{U}(p)$,
- (f) $(s, t) \in \gamma(a^c)$ iff $(t, s) \in \gamma(a)^{-1}$.

Kozen and Parikh proceed to show that if q is a non-theorem then $\mathfrak{S}$ invalidates q . This method is similar to the approach we took above. In this section we present an example of a Segerberg model, which we call the Kozen-Parikh Model (since it is isomorphic to the model $\mathfrak{S}$).

Notice that if $p \in F$ and $s \in \text{At}_F$ then either $\{s\} \vdash_M p$ or $\{s\} \vdash_M \neg p$, since p or $\neg p$ must appear in the conjunction s . Thus, since each atom of F is also M -consistent, then, in a sense, it is maximal modulo F (just as the states of the Fischer-Ladner filtrations of canonical models are maximal modulo F). The next few lemmas show this observation from a slightly different perspective. As usual, let $\mathfrak{C}_M = (U_M, \pi_M, \nu_M)$ and $\mathfrak{C}_M^F = (U_M^F, \pi_M^F, \nu_M^F)$ be the canonical model of M and Fischer-Ladner filtration of $\mathfrak{C}_M$ through F .

4.4.2 Lemma: Let $s \in \text{At}_F$ and $u \in U_M$ then, $s \in u$ iff $\forall w \in |u|, s \in w$.

Proof: Let $s = s_1 \& \dots \& s_n$ and suppose that $w \approx_F u$. Since $\text{Max}_M u$, $s \in u$ iff, for $1 \leq i \leq n$, $s_i \in u$. Therefore, we are only required to show that $s_i \in u$ iff $s_i \in w$, which is immediate from the definition of $w \approx_F u$. ■

4.4.3 Lemma: (a) $\forall s \in \text{At}_F, \exists u \in U_M$ such that $s \in u$;

(b) $\forall u \in U_M, \exists s \in At^F$ such that $s \in u$.

Proof:

(a) Let $s = s_1 \& \dots \& s_n \in At^F$. Since $\{s_i \mid 1 \leq i \leq n\}$ is M -consistent, we are assured by Lindenbaum's lemma a $u \in U_M$ with $s \in u$.

(b) Let $u \in U_M$. Consider the set $S = u \cap (F \cup \neg F)$ where $\neg F = \{\neg q \mid q \in F\}$. Since $\text{Max}_M u$ then for all $q \in F$ either $q \in S$ or $\neg q \in S$ (but not both). Also, since u is M -consistent then so is S . Let s be the conjunction of the elements of S (properly ordered by index according to (1)) then $s \in At^F$. ■

Let $u, v \in U_M$ and let s^u and s^v denote the respective atoms of F guaranteed by 4.4.3b (i.e., $s^u \in u$ and $s^v \in v$). We are assured by 4.4.2 that $s^u = s^v$ if and only if $|u| = |v|$. If we let $f(|u|) = s^u$ then f is a bijective mapping from U_M^F to At_F . (That f is surjective follows from 4.4.3a.) Thus, incidentally, the size of U_M^F is the same as the size of At_F . If F has n members then At_F can have at most 2^n members.

4.4.4 Lemma: For all $u \in U_M$ and all $p \in F$, $\{s^u\} \vdash_M p$ if and only if $\mathcal{C}_M^F \Vdash_{|u|} p$.

Proof: Note first that, since $\text{Max}_M u, p \in u$ iff $u \vdash p$ (2.6.1a) iff $\mathcal{C}_M^F \Vdash_{|u|} p$ (by the Filtration theorem). If $\{s^u\} \vdash p$ then $u \vdash p$ (since $s^u \in u$). Conversely, if $p \in u$ then, because $p \in F$, then $p \in u \cap F$ and p must be in the conjunction s^u . Thus $\vdash s^u \rightarrow p$ and finally $u \vdash p$. ■

Define the Kozen-Parikh model for q to be the model $\mathfrak{R} = (W, \xi, \tau)$ where, for $A \in \Pi_0$ and $P \in \Sigma_0$,

$$\begin{aligned} W &= U_M^F \\ (|u|, |v|) \in \xi(A) &\text{ iff } (s^u, \langle A \rangle s^v) \text{ is } M\text{-consistent} \\ \text{for } P \in F, \tau(P) &= \nu_M^F(P) \\ \text{for } P \notin F, \tau(P) &= \emptyset. \end{aligned}$$

and where ξ is extended to a standard Kripke model (in the usual way) to cover the case of compound programs, i.e., let $a, b \in H$ and $|u|, |v| \in W$,

- (a)' $(|u|, |v|) \in \xi(a+b)$ iff $(|u|, |v|) \in \xi(a) \cup \xi(b)$,
- (b)' $(|u|, |v|) \in \xi(ab)$ iff $(|u|, |v|) \in \xi(a) \circ \xi(b)$,
- (c)' $(|u|, |v|) \in \xi(a^*)$ iff for some $k \leq \omega$, $(|u|, |v|) \in \xi(a)^k$,
- (d)' $(|u|, |v|) \in \xi(\lambda)$ iff $|u| = |v|$.

- (e)' $(|u|, |v|) \in \xi(p?)$ iff $|u| = |v|$ and $|u| \in \tau(p)$,
 (f)' $(|u|, |v|) \in \xi(a^c)$ iff $(|v|, |u|) \in \xi(a)^{-1}$.

Notice that $\mathfrak{R}$ closely resembles the model $\mathfrak{S}$ defined above. Indeed, $\mathfrak{R}$ is motivated by the 1-1 correspondence f and 4.4.4. We now demonstrate that $\mathfrak{R}$ is a Segerberg model of $\{q\}$. Thus we are only required to show that $\mathfrak{R}$ is a finest Fischer-Ladner filtration through $\{q\}$ of $\mathfrak{C}_M$. To this end, since $W = U_M^F$ and, for any $P \in F \cap \Sigma_0$, $\tau(P) = \nu_M^F$, we must only demonstrate that ξ satisfies the condition of the next lemma.

4.4.5 Lemma: For all $A \in \Pi_0$ with $\langle A \rangle p \in F$ for some $q \in F$,

$(|u|, |v|) \in \xi(A)$ iff $\exists u', v' \in U_M$ such that $u \simeq_F u'$, $v \simeq_F v'$ and $(u', v') \in \pi_M(A)$.

Proof: Suppose that $(u, v) \in \pi_M(A)$. This assumption is synonymous with the condition that, if $p \in v$ then $\langle A \rangle p \in u$. Thus, since $s^v \in v$ then $\langle A \rangle s^v \in u$. Because $\text{Max}_M u$ and $s^u \in u$ it follows that $\{s^u, \langle A \rangle s^v\}$ is M -consistent, i.e., $(|u|, |v|) \in \xi(A)$.

Conversely, suppose that $(|u|, |v|) \in \xi(A)$, i.e., $\{s^u, \langle A \rangle s^v\}$ is M -consistent and so, according to 4.4.1, we have $s^u = s^v$. We shall construct $w \in U_M$ such that $w \simeq u$, $w \simeq v$ and $\langle A \rangle p \mid p \in w \subseteq w$. Let MD be the logic M supplemented by the rule $\vdash_{MD} q$ implies $\vdash_{MD} \langle A \rangle q$. We know this logic is consistent since it is a subset of the consistent logic $D?c$ (3.2.17 and 4.1.14). Now $\{s^u, \langle A \rangle s^v\}$ is MD -consistent, therefore, by Lindenbaum's lemma, there is a maximal extension of $\{s^u, \langle A \rangle s^v\}$, say w , i.e., $\text{Max}_{MD} w$. Also $\text{Max}_M w$ since $M \subseteq MD$. Now set $u' = v' = w$. Clearly, $\langle A \rangle p \mid p \in v' \subseteq u'$ since u' is closed under the supplementary rule. Also $s^u \in u'$ and $s^v \in v'$ (since $s^u = s^v$ by 4.4.1). Thus, according to 4.4.2, $u' \simeq u$ and $v' \simeq v$ respectively. ■

Chapter 4 Notes:

canonical history

Canonical models for propositional modal logic were invented independently by Dana Scott (in [LEM77]) and David Makinson (in *On some completeness theorems in modal logic*, *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, Vol.12 (1966), pp.379-384). The development of canonical models brought about a significant relief from the cumbersome and difficult tableau methods of Kripke's earlier completeness results in modal logic. The new technique brought together "Kripke's notion of accessibility [$\pi(a)$] ...", Carnap's explication of a possible world [state] as a set of sentences of a certain kind, and Lindenbaum's theorem [theorem 2.6.2] ..." [BOO79;pp.89]. Canonical models of propositional dynamic logics are simple extensions of the ones used for conventional modal logics. However, whereas completeness results for normal propositional modal logics are nearly always straightforward from the canonical construction, this is not the case for PDLs. For the completeness of PDLs with respect to standard Kripke models, proofs are more complicated requiring further filtration. This is because canonical models may be non-standard Kripke models, i.e., they may not interpret a^* as giving the reflexive transitive closure.

canonical completeness proofs

The completeness proofs [PAR78], [SEG78], [K&P81], [HAR84] and the unpublished sketch by Gabbay are undertaken with respect to SKM only. They are basically the same as ours: they all require the construction of a canonical model with an additional Fischer-Ladner filtration. In [PAR78], Parikh uses a tableau construction to get the initial "canonical" model and then he applies filtration. [K&P81] and [HAR84] combine the canonical construction with a filtration in a single step (see section 4.4 where we show how these proofs are related to the one given here).

The completeness result sketched out in [PRA78] is a Gentzen type proof. More replete completeness proofs for sequent type PDLs are given by [NIS79] and [LEI81].

extension of canonical models

Clause (d) of the above definition extends the usual definition of canonical model (e.g., [SEG78]) to deal with the *repeat* operator. I am not aware of any published work which directly extends canonical models in a similar way.

Seegerberg's example

This example is from [SEG78;pp.39]. (The reader will note that the models N_n were used in chapter three (section 3.3) to demonstrate the non-compactness of SKM).

completeness of PDLs with loop

PDL P?1 (P? supplemented by *loop* - instead of *repeat*) has a simple and complete axiomatization. This result was announced by A. Pnueli and R.

Sherman (*Propositional Dynamic Logic of Looping Flowcharts*, Technical Report, Weizman Institute, Rehovot, Israel, 1983). However, their techniques do not seem to carry over to Pr (reported in [H&S82;pp.190] and [H&S85;pp.134]).

Harel's proof

A similar proof is outlined by Harel in [HAR84;pp.510-515]. The result is due to Harel and Sherman and combines the ideas of [K&P81] with the decidability result of Pratt [PRA79a]. The original proof was flawed but was corrected, and reapplied in a slightly different context in [H&S85] where "regular programs" are replaced by finite state machines.

atom

"The term *atom* is from Boolean algebra: if $\mathbf{B}$ is the Boolean algebra of formulas of $\mathbf{P}$ modulo provable equivalence, then the [elements of $\text{At}_{\mathbf{F}}$] are atoms of the Boolean subalgebra of $\mathbf{B}$ generated by the elements of $\mathbf{F}$." [K&P81; pp.115, note 1] Clearly, the number of atoms of $\mathbf{F}$ is not more than 2^n where n is the cardinality of $\mathbf{F}$.

CHAPTER FIVE

COMPARING EXPRESSIVE POWER

So far languages have only been distinguished by the most superficial of syntactic comparisons (Chapter two, section 2.2). However, real expressive power is derived from semantics. And so, in the case of dynamic logics, expressive power is measured by the ability to distinguish between different states of a model. In this section we demonstrate how each of the enhancements actually increase the expressive power of propositional dynamic logic. In the process many examples of standard Kripke models are given.

Let L_1 and L_2 be two PDLs whose expressions are from $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively (and in particular, from Σ_1 and Σ_2 respectively). Let p_1 and p_2 be well-formed formulae of $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively and let $\mathfrak{S}=(W,\pi,\nu)$ be a standard Kripke model. (In this section we shall restrict our attention to standard Kripke models since these are the most "intuitive" or familiar - albeit non-compact - structures of before/after reasoning about programs.) We say that p_1 and p_2 are semantically equivalent in $\mathfrak{S}$ (i.e., have the same meaning for $\mathfrak{S}$) if, for all $w \in W$, $\mathfrak{S} \models_w p_1$ if and only if $\mathfrak{S} \models_w p_2$. Also, p_1 and p_2 are said to be semantically equivalent if they are semantically equivalent in all standard Kripke models. L_1 has the same expressive power to L_2 if and only if every formula of Σ_1 is semantically equivalent to some formula of Σ_2 and vice versa. L_2 has more expressive power than L_1 , and is denoted $L_1 < L_2$, if there is a formula of Σ_2 and a standard Kripke model $\mathfrak{S}$ for which there is no semantically equivalent Σ_1 formula. Note that even if $L_1 < L_2$, it is possible for $L_2 < L_1$ but this would require that neither Σ_1 or Σ_2 be subsets of the other.

In what follows below we shall liberally apply the soundness results of chapter

three. Application of soundness will be implicit when we take recourse to a theorem of the logic under consideration.

5.1 The Expressive Power of Tests

That tests add to the expressive power of P is shown in [B&P81] and sketched in [HAR84;pp.526-527]. Here we basically report the proof of [B&P81] and add the proof of lemma 5.1.1 from [GIN66;pp.86]. The elementary version of PDL in [B&P81] is, however, slightly different from the logic P . [B&P81] allow the empty program and eliminate λ . This formulation impacts only on the proof of the reduction lemma 5.1.2. We also demonstrate that P_c , P_r and P_{cr} are less expressive than $P?$, thereby adding to the results of [B&P81].

Since $Ax.;$, $Ax.+$ and $Ax.?$ provide for the elimination of tests which are not bound to the $*$ -operator, we should expect that tests add to the expressive power of PDL by virtue of "test iteration". We shall demonstrate that no formula of Σ_{PDL} is semantically equivalent to the $\Sigma_{PDL,?}$ formula

$$t = \langle (P?A)^*(\neg P?)A \rangle P$$

for any $P \in \Sigma_0$ and $A \in \Pi_0$ for some standard Kripke model.

The proof that logic $P?$ is more expressive than P uses a type of Cyclic standard Kripke model which we call a Berman-Paterson model [B&P81]. For $m > 1$ and $A \in \Pi_0$, a Berman-Paterson model (figure 5-1) $\mathcal{C}(A, m) = (W, \pi, \nu)$ is one where

$$\begin{aligned} W &= \{0, 1, \dots, 2m\} \\ \pi(A) &= \{(i, i+1) \mid i=0, 1, \dots, 2m-1\} \cup \{(2m, 0)\} \\ \pi(B) &= \emptyset \text{ for } B \in \Pi_0 - \{A\} \\ \nu(P) &= W - \{m-1, 2m-1, 2m\}. \end{aligned}$$

$\mathcal{U}(A, m)$

Berman-Paterson model

figure 5-1

Let $A \in \Pi_0$ and let $\Pi^A \subseteq \Pi_{PDL}$ denote the set of regular expressions over A , i.e., Π^A is the set of all $\mathcal{L}_{PDL}$ programs made up from the basic program A only. As a regular expression, $a \in \Pi^A$ determines a regular set or language (i.e., one accepted by a finite automaton {NOTE: finite automaton}). The regular set determined by $a \in \Pi^A$ can be thought of as the set of all execution sequences of A that satisfy the program a . Thus, for example, the regular set determined by the program $a = A + AA + A\{AAA\}^*$ is the set $\{A, AA, AAA, AA^3, AA^6, \dots\}$. The regular set determined by the program $a = A^{i+j} + A^k\{A^l\}$ is the set $\alpha = \{A^{i+j}, A^{i+2j}, \dots, A^{k+l}, A^{k+2l}, \dots\}$. Notice that where two different programs, say a and b , determine the same regular set then the formulae $\langle a \rangle p$ and $\langle b \rangle p$ are semantically equivalent. An interesting feature of regular sets (and indeed, any context-free language) over a singleton alphabet is that they are all eventually periodic. Thus, the set α is also determined by a Π^A program of the form $b + c(A^n)^*$. This is the point of the next lemma.

5.1.1 Lemma: Let $a \in \Pi^A$ then there exists a $d \in \Pi^A$ of the form $b + c(A^n)^*$ where b and c are $*$ -free expressions from Π^A (possibly empty strings) and n is some non-zero integer such that a and d determine the same regular set.

Proof: Let $\alpha = \{A^i | i \in X\}$ be the regular set determined by a (i.e., X is a subset of the positive integers which gives the number of A 's in the string). If $z \in \alpha$ then let $/z/$ denote the length of z (i.e., number of A 's in z). The proof requires the well known "uvwxy theorem" of context-free languages. {NOTE: uvwxy theorem} This theorem provides integers h and m such that for each $z \in \alpha$ where $/z/ > h$,

$$\begin{aligned} & z \text{ is of the form } uvwxy, \\ & \text{not } /uv/ = 0 \text{ (i.e., } uv \text{ is not the empty string),} \\ & /uvw/ \leq m, \\ & xu^k w v^k y \in \alpha, \text{ for all } k \geq 1. \end{aligned}$$

Clearly, $z = xwyuv = xwyA^j$ for some $j \leq m$, and $xwy(A^j)^k \in \alpha$ for all $k \geq 1$. Let $n = m!$ (m factorial), then for any $z \in \alpha$ where $/z/ > h$ the regular set determined by $z(A^n)^*$ is a subset of α . Now, for each i , $1 \leq i \leq n$, let α_i be the regular set determined by the regular expression $A^{h+i}(A^n)^* \cap \alpha$. When α_i is not empty, let $\mu \alpha_i$ denote the shortest member of α_i , i.e., $\forall z \in \alpha_i, / \mu \alpha_i / \leq /z/$. When α_i is empty let $\mu \alpha_i$ be undefined. Let $c = \cup_{i=1}^n \mu \alpha_i$.

Clearly, each $z \in \alpha$ with $|z| > h$ is in the regular set determined by $c(A^n)^*$. Let b be the regular expression which determines the finite set $\{z \in \alpha \mid |z| \leq h\}$. Then α is determined by the regular expression $b+c(A^n)^*$. ■

5.1.2 Reduction Lemma for P: For any formula $p \in \Sigma_{PDL}$ there is a formula $q \in \Sigma_{PDL}$ that is semantically equivalent to p in $\mathcal{C}(A, m) = (W, \pi, \nu)$ for all $m > 0$ such that:

- (a) A is the only basic program occurring in q ,
- (b) if $\langle a \rangle r \in \text{SUBF}(q)$ then $a = A$ or $a = (A^n)^*$.

Proof: For (a) notice that, since $\pi(B) = \emptyset$ for any basic program B other than A , then, for any program a , by replacing any occurrence in p of aB , Ba or B^* with B , any occurrence in p of $a+B$ or $B+a$ with a , or any subformula of p of the form $\langle B \rangle r$ with $\perp$, then the resulting Σ_{PDL} formula would be semantically equivalent to p in $\mathcal{C}(A, m)$. By carrying out this reduction as far as possible we arrive at a Σ_{PDL} formula whose only basic program is A .

For (b) assume that we have carried out the reduction of (a) and let p_0 denote the resulting Σ_{PDL} formula. Let q be the formula that results from p_0 by replacing all subformulae of p_0 of the form $\langle a \rangle r$ by semantically equivalent formulae of the form $\langle b+c(A^n)^* \rangle r$ where the program $b+c(A^n)^*$ is guaranteed by lemma 5.1.1 and then, by 3.2.4 and 3.2.5, eliminating all occurrences of the $+$ and $;$ -operators from $\langle b+c(A^n)^* \rangle$. Now, any modality occurring in q is either $\langle A \rangle$ or $\langle (A^n)^* \rangle$ for some $n \geq 1$. ■

5.1.3 Lemma: Let $p \in \Sigma_{PDL}$ in which the only modalities occurring are $\langle A \rangle$ or $\langle (A^n)^* \rangle$ for some $n \geq 1$ and let m and k be integers such that $m > 1$, $0 \leq k \leq m$ and $2m+1$ is a prime. If the number of occurrences of the modality $\langle A \rangle$ in p is strictly less than $m-k$ then

$$\mathcal{C}(A, m) \models_k p \text{ if and only if } \mathcal{C}(A, m) \models_{k+m} p.$$

Proof: Let $|q|_{\langle A \rangle}$ denote the number of occurrences of $\langle A \rangle$ in a formula q . The proof is based upon induction on the complexity of $p \in \Sigma_{PDL}$:

Case 1- $p = P \in \Sigma_0$: Recall that $\nu(P) = W - \{m-1, 2m-1, 2m\}$ and observe that the only value for k where not $\mathcal{C}(A, m) \models_k P$ corresponds to $k+m=2m-1$. The only other problematic state is $k=m$, but in this case $|P|_{\langle A \rangle}$ must be strictly less than 0!

Case 2- $p = \neg q$: Clearly $|q|_{\langle A \rangle} = |p|_{\langle A \rangle}$, so we have, by the induction hypothesis, $\mathcal{C}(A, m) \models_k q$ iff $\mathcal{C}(A, m) \models_{k+m} q$. Thus, $\mathcal{C}(A, m) \models_k \neg q$ iff $\mathcal{C}(A, m) \models_{k+m} \neg q$ (by

3.1.1).

Case 3- $p=qvr$: Since $/q/_{\langle A \rangle} \leq /p/_{\langle A \rangle}$ and $/r/_{\langle A \rangle} \leq /p/_{\langle A \rangle}$, then by the induction hypothesis, $\mathcal{C}(A, m) \models_k q$ iff $\mathcal{C}(A, m) \models_{k+m} q$ and $\mathcal{C}(A, m) \models_k r$ iff $\mathcal{C}(A, m) \models_{k+m} r$. Therefore, by 3.1.1, $\mathcal{C}(A, m) \models_k qvr$ iff $\mathcal{C}(A, m) \models_{k+m} qvr$.

Case 4- $p=\langle A \rangle q$: Note that $/q/_{\langle A \rangle} = (/p/_{\langle A \rangle} - 1) < m - (k+1)$. Now, $\mathcal{C}(A, m) \models_k \langle A \rangle q$ iff $\mathcal{C}(A, m) \models_{k+1} q$ (by 3.1.1d) iff $\mathcal{C}(A, m) \models_{k+1+m} q$ (by the induction hypothesis) iff $\mathcal{C}(A, m) \models_{k+m} \langle A \rangle q$ (by 3.1.1d).

Case 5- $p=\langle A^n \rangle^* q$: Let n modulo $(2m+1) = d$.

If $d=0$ then $(k, j) \in \pi(A^n)$ iff $k=j$. Hence, $\mathcal{C}(A, m) \models_k \langle A^n \rangle^* q$ iff $\mathcal{C}(A, m) \models_k q$ (by 3.1.1d). The required result is then immediate by the induction hypothesis. Now, if not $d=0$ then, because $2m+1$ is prime, (ni) modulo $(2m+1)$ every $i \in W$ is different. (NOTE: least residues lemma) Thus, $(k, i) \in \pi((A^n)^*)$ for all $i \in W$ and so, according to 3.1.1d, $\mathcal{C}(A, m) \models_k \langle A^n \rangle^* q$ iff $\mathcal{C}(A, m) \models_{k+m} q$.

■

Thus, by carefully choosing m , it is impossible for a Σ_{PDL} formula that contains only the modalities $\langle A \rangle$ and $\langle A^n \rangle^*$ to distinguish between the "bottom or top-hand" portion of $\mathcal{C}(A, m)$ (see figure 5-1). Indeed, inside a loop it is always possible to proceed from any state to any other state. On the other hand, the ability to test atomic P inside of a loop enables evaluation at any state. Consequently P is less expressive than $P?$.

5.1.4 Theorem: $P < P?$.

Proof: Suppose that $p \in \Sigma_{PDL}$ is semantically equivalent to the formula t and let q be the Σ_{PDL} formula semantically equivalent to p in $\mathcal{C}(A, m)$ (for any m) guaranteed by reduction lemma 5.1.2b. By lemma 5.1.3 we have, for any $m > 1$ with $2m+1$ being prime and by setting $k=0$,

$$\mathcal{C}(A, m) \models_0 q \text{ iff } \mathcal{C}(A, m) \models_m q.$$

This result is obviously false in consideration of $\mathcal{C}(A, m)$. Observe that (a) $\mathcal{C}(A, m) \models_0 t$ but not (b) $\mathcal{C}(A, m) \models_m t$. To see these note that, by 3.1.1, the conditions of standard Kripke models and the definition of $\mathcal{C}(A, m)$, $\mathcal{C}(A, m) \models_h t$ iff there are $i, j \in W$ and integer n such that $(h, i) \in \pi((P?A)^n)$, $i \in \nu(P)$ and $(i, j) \in \pi(A)$. Now (a) follows by setting

$h=0$, $i=m-1$, $j=m$ and $n=m$. By setting $h=m$, however, the lack of any appropriate value for i and n shows that (b) cannot be true. Consequently there can be no $p \in \Sigma_{PDL}$ which is semantically equivalent to t in $\mathcal{C}(A, m)$. ■

Although the proof of $P < P?$ is behind us, it is not obvious that $P_{cr} < P?$ (and so also $P_c < P?$ and $P_r < P?$). This is due to the fact that with the introduction of converse we can no longer employ lemma 5.1.1 as it stands. However, the structure of Berman-Paterson models still permits us to reduce the problem to one of regular sets over a singleton alphabet. We use the fact that $\Pi_{PDL, c, repeat} \supseteq \Pi_{PDL, c}$ is the set of regular expressions over the alphabet $\lambda \cup \Pi_0 \cup \{c\}$ and that theorems 2.5.13 to 2.5.16 allow us to restrict the converse operator to basic programs.

5.1.5 Reduction lemma for P_{cr} : For any formula $p \in \Sigma_{PDL, c, repeat}$ there is a formula $q \in \Sigma_{PDL, c, repeat}$ that is semantically equivalent to p in $\mathcal{C}(A, m) = (W, \pi, \nu)$ for all $m > 0$ such that:

- (a) A is the only basic program occurring in q ,
- (b) if $\langle a \rangle r \in SUBF(q)$ then $a = A$ or $a = (A^n)^*$,

Proof: For (a) we need only supplement the replacement rule of proof 5.1.2a with the following: Let a be any program. If B is any basic program other than A replace any occurrence in p of B^c , $(aB)^c$ or $(Ba)^c$ with B , any occurrence in p of $(a+B)^c$ or $(B+a)^c$ with a^c , or any subformula of p of the form $repeat(B)$ with $\perp$.

For (b) assume that we have carried out the reduction of (a) and let p_1 denote the resulting $\Sigma_{PDL, c, repeat}$ formula. Assume that the converse operator is only directly attached to the atomic program A (by virtue of 2.5.13 through 2.5.16). Because $\mathcal{C}(A, m)$ is cyclic in nature (modulo $2m+1$) we can replace any occurrence in p_1 of $(A^c)^j$ for any $j > 0$ with A^n where $n = (-j) \text{ modulo } (2m+1)$. Accordingly, we reduce p_1 to a semantically equivalent formula p_0 that does not contain an occurrence of the converse operator. (Note that the formula that results from this reduction is of $\Sigma_{PDL, repeat}$.) The remainder of the proof follows that of lemma 5.1.1. ■

In order to complete the proof we need an addendum to lemma 5.1.3 to take care of the case where p is a formula of the type $repeat(a)$. Actually, a stronger result is

transparent: $\text{repeat}(a)$ is globally true for any program $a \in \Pi^A$.

5.1.6 Lemma: Let $a \in \Pi^A$. $\mathcal{C}(A, m) \models \text{repeat}(a)$.

Proof: The regular set determined by $a \in \Pi^A$ contains only strings of A's of finite length. The cyclic nature of $\mathcal{C}(A, m)$ guarantees that, regardless of its starting state, any execution of A^i (for any $i > 0$) will bring about a state from which A^j (for any $j > 0$) may execute again, i.e., $v(\text{repeat}(a)) = W$, for any $a \in \Pi^A$. ■

5.1.7 Theorem: $P_c, P_r, P_{cr} < P?$

Proof: As in the proof of 5.1.4, using lemma 5.1.5 instead of 5.1.2, and supplementing 5.1.3 with 5.1.6 (and replace Σ_{PDL} with $\Sigma_{PDL, c, \text{repeat}}$). ■

We can refine the above result to obtain a much stronger one - and a finer grained understanding of the supplementary expressive power of propositional dynamic logic with tests. (NOTE: finer grained result) Notice that in formula t tests are applied only to the atomic proposition P . Therefore, the above analysis has shown that we add to the expressive power of P by simply allowing the testing of atomic propositions. Furthermore, the formula t does not make use of nesting tests within programs allowed in $\Sigma_{PDL, ?}$. Let $\Sigma_{PDL}^0 = \Sigma_{PDL}$ and let Σ_{PDL}^{n+1} be the subset of $\Sigma_{PDL, ?}$ in which programs are allowed to contain the test $p?$ only for $p \in \Sigma_{PDL}^n$ (- the superscript refers to the maximum level of nesting allowed). Refer to the resulting logics by $P^0 (= P), P^1, \dots$ (where for $i > 0$, P^i is supplemented by $Ax?$). According to this hierarchy, t is obviously a Σ_{PDL}^1 formula. It turns out that each increment of the allowed level of nesting of tests increases expressive power. The method employed to show this is a generalization of one used above which we now sketch. Define, recursively for each level n of the hierarchy, a Σ_{PDL}^n formula t_n :

$$\begin{aligned} t_1 &= \langle A_0(P?A_0)^*(\neg P?)A_0 \rangle P \\ t_{n+1} &= \langle A_n(t_n?A_n)^*(\neg t_n?)A_n \rangle P \end{aligned}$$

where $A_i \in \Pi_0$. Structures, which are essentially layers of simple Berman-Paterson models, are used to show that no Σ_{PDL}^n formula is semantically equivalent to t_{n+1} . Let

$\mathcal{C}^n(A_n, m_n)$ be the Berman-Paterson model at level n . Accessibility between different levels of the layered structure are confined to adjacent levels only. And, it is only through the atomic program A_i that levels i and $i+1$ connected: Each state of level $i+1$ goes to one of two distinguished states of level i , specifically state 1 and state m_i+1 of level i . (See figure 5-2 for a pictorial representation of these layered Berman-Paterson models.) This accessibility constraint ensures that satisfaction of a test in the nesting corresponds to descending a level in the multi-layered model. $P^i < P^{i+1}$ is proved by showing that states 1 and m_i+1 of level i can only be distinguished by formulae which trace a path to the 0th level of the layered model, where they may be distinguished by a forced evaluation of P . Such formulae must have nested tests to a depth of at least $i+1$, such as the formula t_{i+1} - thus proving that ascending levels of the PDLs P^j gain expressive power.

5.2 The Expressive Power of Reverse Computations

A sketch of the proof that logic P_c is more expressive than $P?$ is given in [HAR84; pp.538-539]. The proof is straightforward, and we extend it to show that the inverse operator adds expressive power to all the PDLs under our consideration.

5.2.1 Theorem: $P?r < P_c$.

Proof: Consider the following standard Kripke model $\mathcal{C} = (W, \pi, \nu)$ where

$$W = \{w, x, y\},$$

$$\forall P \in \Sigma_0, \nu(P) = \{w, y\},$$

$$\forall A \in \Pi_0, \pi(A) = \{(x, y)\}.$$

Thus, state w is isolated - there are no A transition leading into or out of w . Clearly $\mathcal{C} \models_y \langle A^c \rangle \top$ but not $\mathcal{C} \models_w \langle A^c \rangle \top$. We now show, by induction on the complexity of $p \in \Sigma_{PDL, \tau, \text{repeat}}$, that no formula is able to distinguish between the states w and y .
Case 1 - $p = P \in \Sigma_0$ or $p = \perp$: It is clear that neither $P \in \Sigma_0$ or $\perp$ is able to distinguish

between w and y - their evaluation is the same at both w and y .

Case 2- $p=\neg q$: $\mathcal{C} \models_w q$ iff $\mathcal{C} \models_y q$ by the induction hypothesis, therefore $\mathcal{C} \models_w p$ iff $\mathcal{C} \models_y p$ by 3.1.1b.

Case 3- $p=q \vee r$: $\mathcal{C} \models_w q \vee r$ iff $\mathcal{C} \models_w q$ or $\mathcal{C} \models_w r$ (by 3.1.1c) iff $\mathcal{C} \models_y q$ or $\mathcal{C} \models_y r$ (by the induction hypothesis) iff $\mathcal{C} \models_y q \vee r$ (by 3.1.1c).

Case 4- $p=\text{repeat}(a)$: Notice that, for all such formulae not $\mathcal{C} \models_u p$ for any $u \in W$.

Case 5- $p=\langle a \rangle q$: By induction on the complexity of $a \in \Pi_{\text{PDL}, ?, \text{repeat}}$.

Case 5.1- $a=A$, $A \in \Pi_0$ or $a=\lambda$: If $A \in \Pi_0$ then clearly not $\mathcal{C} \models_y \langle A \rangle q$ and not $\mathcal{C} \models_w \langle A \rangle q$ since $(y, u), (w, u) \notin \pi(A)$ for any $u \in W$ (i.e., $\langle A \rangle q$ cannot distinguish states w and y in $\mathcal{C}$). Where $a=\lambda$ the proof is trivial.

Case 5.2- $a=b+c$: $\mathcal{C} \models_y \langle b+c \rangle q$ iff $\mathcal{C} \models_y \langle b \rangle q$ or $\mathcal{C} \models_y \langle c \rangle q$ (by $\text{Ax.}+$ and 3.1.1c) iff $\mathcal{C} \models_w \langle b \rangle q$ or $\mathcal{C} \models_w \langle c \rangle q$ (by the induction hypothesis) iff $\mathcal{C} \models_y \langle b+c \rangle q$ (by 3.1.1c and $\text{Ax.}+$).

Case 5.3- $a=bc$: With the possible exception of y itself there is no other $u \in W$ such that $(y, u) \in \pi(c)$. Consequently $\langle a \rangle q$ is semantically equivalent to either $\perp$ (case 1) or $\langle b \rangle q$ in $\mathcal{C}$, and by the induction hypothesis $\langle b \rangle q$ cannot distinguish states w and y .

Case 5.4- $a=b^*$: Observe that $\langle b^* \rangle q$ is semantically equivalent to $\langle b^2 \rangle q$ in $\mathcal{C}$, which is just a special case of 5.3.

Case 5.5- $a=r?$: Using $\text{Ax.}?$ p is semantically equivalent to the formula $q \& r$ in $\mathcal{C}$, which is just case 3.

■

5.2.2 Corollary: $P, P?, Pr < P?c, Pcr, P?cr$.

5.3 The Expressive Power of Infinite Computations

The ground work for this section was done in chapter three (section 3.4). We have seen that the *repeat* operator induced failure of the collapsed SKM-model property (theorem 3.4.5) and that, in tandem with the converse operator, it prevented the finite

SKM-model property (theorem 3.4.7).

5.3.1 Theorem: Let L be a PDL that satisfies the collapsed SKM-model property, then $L < Pr$.

Proof: Since Pr does not have the collapsed SKM-model property, there is a $p \in \Sigma_{PDL, repeat}$ which is satisfied (by $\mathfrak{S}$, say) which is not satisfiable by the quotient model $\mathfrak{S}/FLC((p))$. Now since $\mathfrak{S}/FLC((p))$ is a standard Kripke model (theorem 3.4.1) and since any formula of L which is satisfiable in $\mathfrak{S}$ is also satisfiable in $\mathfrak{S}/FLC((p))$ then there is no formula of L which is semantically equivalent to p . ■

5.3.2 Corollary: $P, P?, Pc, P?c < Pr, P?r, Pcr, P?cr$.

Proof: $P, P?, Pc, P?c$ all satisfy the collapsed SKM-model property (theorem 3.4.4) and $Pr \not\subset P?r, Pcr$ and $P?cr$. ■

5.3.3 Theorem: Let L be a PDL that satisfies the finite SKM-model property, then $L < Pcr$.

Proof: Since Pr does not satisfy the finite SKM-model property, there is a satisfiable formula $p \in \Sigma_{PDL, c, repeat}$ which is not satisfied by any finite standard Kripke model. Since any satisfiable formula of L is satisfiable in some finite standard Kripke model, there is no formula of L which is semantically equivalent to p . ■

5.3.4 Corollary: $Pr, P?r < Pcr, P?cr$.

Proof: Pr, Pcr satisfy the finite SKM-model property (theorem 3.5.?) and since $Pcr \not\subset P?cr$, then $P?cr$ is more expressive than Pr and $P?r$. ■

Of course, this last result implies that infinite memory permits the addition of expressive power to PDLs - which is not a particularly surprising result. (NOTE: comparison with *loop*)

5.4 Summary of Results on Expressive Power

Essentially we have shown that any PDL which supports one of the $?$, c or *repeat* operators is more expressive than a PDL that lacks it. Thus

$P < P?, Pc, Pr, P?c, P?r, Pcr, P?cr;$

$P? < Pc, Pr, P?c, P?r, Pcr, P?cr;$

$Pc < P?, Pr, P?c, P?r, Pcr, P?cr;$

$Pr < P?, Pc, P?c, P?r, Pcr, P?cr;$

$P?c < Pr, P?r, Pcr, P?cr;$

$P?r < P?, P?c, Pcr, P?cr;$

$Pcr < P?, P?c, P?r, P?cr.$

Chapter 1 Notes:

finite automaton

See theorem 3.10 pp.38-39 of [H&U69;pp38-39] or theorem 2.1.1b of [GIN66;pp.48].

uvxyz theorem

Note that all regular sets are context free languages. The proof of the uvxyz theorem is beyond the methods and terminology developed so far. Nor is this theorem of direct interest to us. See lemma 3.1.1 of [GIN66;pp.84] or theorem 4.7 of [H&U69;pp.57-59].

least residues lemma

For a proof of this refer to lemma 1, pp.44 of Underwood Dudley's *Elementary Number Theory*, 2nd Edition, W.H. Freeman and Company, San Francisco.

finer grained result

The results of this paragraph are sketched in [B&P81;pp.326-328] and were arrived at independently by Berman (in *Expressiveness Hierarchy for PDL with rich tests*, Technical Report #78-11-10, University of Washington, Seattle, Washington (1978)) and Paterson (in *The Power of Tests in Propositional Dynamic Logic*, Technical Report #47, University of Rochester, Rochester, New York).

comparison with loop

If *repeat* is exchanged with the *loop* operator discussed in the introduction, the same results follow - this despite the fact that the *loop* operator is strictly less expressive than *repeat* [H&S82].

BIBLIOGRAPHY

Note:

The bibliography is arranged in alphabetical order on the reference label (e.g. [KRI63]) rather than author.

Abbreviations:

AInf	Acta Informatica
CALP	Colloquium on Automata, Languages and Programming
FofCT proc.	Foundations of Computation Theory, Proceedings
FPC	Formalization of Programming Concepts, International Colloquium
I&C	Information and Control
IBM	IBM Research Report, Yorktown Heights, New York
JCSS	Journal of Computer and Systems Science
JPL	Journal of Philosophical Logic
JSL	Journal of Symbolic Logic
LCS	Laboratory for Computer Science, M.I.T.
LP proc.	Logics of Programs, Proceedings
LNCS	Lecture Notes in Computer Science, Springer-Verlag
MFCS proc.	Mathematical Foundations of Computer Science, Proceedings
MST	Mathematical Systems Theory
SCT	Symposium, Computation Theory
SFCS	Symposium on Foundations of Computing Science
SPPL	Symposium on Principles of Programming Languages
STOC	Symposium on the Theory of Computing
TCS	Theoretical Computer Science

Bibliography:

- [AB79] ABRAHAMSON, Karl, *Modal Logic of concurrent nondeterministic programs*, **Semantics of Concurrent Computation, Proceedings 1979**, LNCS 70, pp.21-33.
- [AHU74] AHO, A.V., HOPCROFT, J.E. & ULLMAN, J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley series in Computer Science and Information Processing (1974).
- [B&P81] BERMAN, F. & PATERSON, M., *Propositional dynamic logic is weaker without tests*, TCS, Vol.16 (1981) pp.321-328.
- [B&S69] BELL, J.L. & SLOMSON, A.B., *Models and Ultraproducts: An introduction*, North-Holland, Amsterdam (1969).
- [BER79] BERMAN, F., *A completeness technique for D-axiomatizable semantics*, 11th ACM STOC, **Proceedings 1979**, pp.160-166.
- [BER80] BERMAN, F., *Nonstandard models in Propositional Dynamic Logic, Logics of Programs and their Applications*, **Proceedings 1980**, ed. A. Salwicki, LNCS 148, pp.81-85.
- [BER82] BERMAN, F., *Semantics of looping programs in propositional dynamic logic*, MST, Vol.15 (1982) pp.285-294.
- [BHP81] BEN-ARI, M., HALPERN, J.Y., & PNUELI, A., *Deterministic propositional dynamic logic: finite models, complexity, and completeness*, MIT/LCS/TM-190 (January 1981). (Also JCSS, Vol.25 (1982) pp.402-417.)
- [BOO79] BOOLAS, G., *The Unprovability of Consistency: An Essay in Modal Logic*, Cambridge University Press, Cambridge (1979).
- [C&K73] CHANG, C.C. & KIESLER, H.J., *Model Theory*, North-Holland, Amsterdam (1973).
- [CHE80] CHELLAS, B.F., *Modal Logic (an Introduction)*, Cambridge University Press, Cambridge (1980).
- [CON77] CONSTABLE, R.L. *On the theory of programming logic*, 9th ACM STOC, **Proceedings 1977**, pp.269-285.
- [CUT80] CUTLAND, N.J., *Computability: an introduction to recursive function theory*, Cambridge University Press, Cambridge (1980).
- [DAN84] DANECKI, Ryszard, *Nondeterministic propositional dynamic logic with intersection is decidable* 5th SCT, **Proceedings 1984**, LNCS 208, pp.34-53.

- [EB78] EMDE BOAS, P. VAN, *The connection between modal logic and algorithmic logics*, MFCS proc. 1978, LNCS 64, pp.1-18.
- [F&L77] FISCHER, M.J. & LADNER, R.E., *Propositional modal logic of programs*, 9th ACM STOC, Proceedings 1977, pp.286-294.
- [F&L79] FISCHER, M.J. & LADNER, R.E., *Propositional dynamic logic of regular programs*, JCSS Vol 18 (1979) pp.194-211.
- [GIN66] GINSBERG, Seymour, *The Mathematical Theory of Context Free Languages*, McGraw-Hill, New York (1966).
- [GOL82] GOLDBLATT, R.I., *Axiomatizing the Logic of Computer Programming*, LNCS 130 (1982)
- [GOL86] GOLDBLATT, R.I., Reviews of [F&L77], [K&P81], [PAR78], [PRA76], [SEG78], JSL, Vol.51, No.1 (March, 1986) pp.225-227.
- [H&P78] HAREL, D., & PRAET, V.R., *Nondeterminism in logics of programs (preliminary report)*, 6th ACM Symp. on Principles of Programming Languages, Proceedings 1978, pp.203-213.
- [H&R81] HALPERN, J.Y. & REIF, J.H., *The propositional dynamic logic of deterministic, well-structured programs (extended abstract)*, 22nd IEEE SFCS, Proceedings 1981, pp.322-334. (Also TCS Vol.27 (1983) pp.127-165.)
- [H&S82] HAREL, D. & SHERMAN, R., *Looping vs. repeating in dynamic logic*, I&C, Vol.55 (1982), pp.175-192.
- [H&S85] HAREL, D. & SHERMAN, R., *Propositional dynamic logic of flowcharts*, I&C, Vol.64 (1985), pp.119-135.
- [H&U69] HOPCROFT, J.E. & ULLMAN, J.D., *Formal Languages and their Relation to Automata*, Addison-Wesley series in Computer Science and Information Processing (1969).
- [HAB84] HABASINSKI, Z., *Model theory of propositional logics of programs, some open problems* 5th SCT, Proceedings 1984, LNCS 208, pp.98-110.
- [HAR78] HAREL, D., *Arithmetical completeness in logic of programs*, 5th CALP, Proceedings 1978, LNCS 62, pp.268-288.
- [HAR79] HAREL, D., *First-Order Dynamic Logic*, LNCS 68 (1979).
- [HAR83] HAREL, D., *Recurring dominoes: making the highly undecidable highly understandable (preliminary report)*, FofCT, proc. 1983, ed. Marek Karpinski, LNCS 158, pp.177-194.

- [HAR84] HAREL, D., *Dynamic Logic* Chapter II.10 of **Handbook of Philosophical Logic** Vol. II, D.Reidel (1984) Dordrecht, Holland, pp.497-604.
- [HKP80] HAREL, D., KOZEN, D. & PARIKH, R., *Process logic: expressiveness, decidability, completeness*, 21nd IEEE SFCS, **Proceedings 1980**, pp.129-142.
- [HMP77] HAREL, D., MEYER, A.R. & PRATT, V.R., *Computability and completeness in logic of programs*, 9th ACM STOC, **Proceedings 1977**, pp.261-268. (A revised version is M.I.T., MIT/LCS/TM-97 (February, 1978).)
- [HPS81] HAREL, D., PNUELI, A. & STAVI, J., *Propositional dynamic logic of context-free programs*, 22th IEEE SFCS, **Proceedings 1981**, pp.310-321.
- [HPS81a] HAREL, D., PNUELI, A. & STAVI, J., *Further results on propositional dynamic logic of nonregular programs*, **LP proc. 1981**, ed. Dexter Kozen, LNCS 131, pp. 124-136.
- [K&P81] KOZEN, D. & PARIKH, R., *An elementary proof of the completeness of PDL (note)*, **TCS**, Vol.14 (1981), pp.113-118. (Also IBM RC8097 (September 1980).)
- [K&P83] KOZEN, D. & PARIKH, R., *A decision procedure for the propositional μ -calculus*, **LP proc. 1983**, ed. Edmund Clarke and Dexter Kozen, LNCS 164, pp.313-325.
- [KOZ79] KOZEN, Dexter, *A representation theorem for models of $\ast$ -free PDL*, IBM RC7864 (September 1979). (Also in 7th Int. CALP, **proceedings 1980**, LNCS 85, pp.351-362).
- [KOZ79a] KOZEN, Dexter, *On the duality of dynamic algebras and Kripke models*, IBM RC7893 (October 1979). (Also **LP proc. 1979**, ed. Erwin Engeler, LNCS 125, pp.1-11.)
- [KOZ81] KOZEN, Dexter, *On induction vs. $\ast$ -continuity*, **LP proc. 1981**, ed. Dexter Kozen, LNCS 131, pp.167-176. (Also IBM RC8486 (September 1980).)
- [KOZ82] KOZEN, Dexter, *Results on the propositional μ -calculus*, 9th CALP, **Proceedings 1982**, LNCS 140, pp.348-359.
- [KP83] KOREN, T. & PNUELI, A., *There exist decidable context free propositional dynamic logics*, **LP proc. 1983**, ed. Edmund Clarke and Dexter Kozen, LNCS 164, pp.290-312.
- [KRI63] KRIPKE, S.A., *Semantical analysis of modal logic I: normal modal propositional calculi*, **Zeit. fur Mathematische Logik & Grundlagen der Mathematik**, 9 (1963), pp.67-96.

- [LEI81] LEIVANT, Daniel, *Proof theoretic methodology for Propositional Dynamic Logic*, FPC, **Proceedings 1981**, LNCS 107, Ed. J. Díaz and I. Ramos, pp.356-373.
- [LEM77] LEMMON, E.J. (in collaboration with Dana S. Scott), *The Lemmon Notes: an Introduction to Modal Logic* (Edited by K. Segerberg) **The American Philosophical Quarterly Monograph Series**, (Edited by N. Rescher) No.11, Basil Blackwell, Oxford (1977).
- [LIP77] LIPTON, R.J., *A necessary and sufficient condition for the existence of Hoare logic*, 18th IEEE SFCS, **Proceedings 1977**, pp.1-6.
- [M&D82] MANDERS, K.L. & DALEY, R.F., *The complexity of the validity problem for dynamic logic*, I&C, Vol.54 (1982) pp.48-69.
- [M&P79] MANNA, Z. & PNUELI, A., *The modal logic of programs*, 6th Int. CALP, **Proceedings 1979**, LNCS 71, pp.385-409.
- [M&T81] MEYER, A.R. & TIURYN, J., *A note on equivalences among logics of programs*, LP proc., 1981, ed. Dexter Kozen, LNCS 131, pp.282-299.
- [M&W82] MEYER, A.R. & WINKLMAN, K., *Expressing program looping in regular dynamic logic*, TCS, Vol.18 (1982) pp.301-323.
- [MSM79] MEYER, A.R., STREETT, R.S., MIRKOWSKA, G., *The deducibility problem in propositional dynamic logic*, MIT/LCS/TM-192, (February 1981). (Also LP proc. 1979, ed. Erwin Engeler, LNCS 125, pp.12-22.)
- [NIS79] NISHIMURA, Hirokazu, *Sequential method in propositional dynamic logic*, AInf, Vol.12 (1979) pp.377-400.
- [NIS80] NISHIMURA, Hirokazu, *Descriptively complete process logic*, AInf, Vol.14 (1980) pp.359-369.
- [PAR78] PARIKH, R., *A completeness result for PDL*, MFCS proc., 1978, LNCS 64, pp.403-415.
- [PAR78a] PARIKH, R., *A decidability result for a second order process logic*, 19th IEEE SFCS, **Proceedings 1978**, pp.177-183.
- [PAR81] PARIKH, R. (with KOZEN, D.), *Propositional dynamic logics of programs: a survey*, MIT/LCS/TM-191, (January 1981). (Also LP proc. 1979, ed. Erwin Engeler, LNCS 125, pp.102-144.) Kozen was responsible for the portion on dynamic algebras.
- [PAR83] PARIKH, R., *Propositional dynamic logics of programs: new directions*, FofCT proc. 1983, ed. Marek Karpinski, LNCS 158, pp.347-359.

- [PNU77] PNUELI, A., *The temporal logic of programs*, 18th IEEE SFCS, Proceedings 1977, pp.46-57.
- [PNU79] PNUELI, A., *The temporal semantics of concurrent programs*, Semantics of Concurrent Computation, Proceedings 1979, LNCS 70, pp.1-20.
- [PRA76] PRATT, V.R., *Semantical considerations on Floyd-Hoare logic*, MIT/LCS/TR-168 (September 1976). (An abbreviated version appears in 17th IEEE SFCS, Proceedings 1976, pp.109-121).
- [PRA78] PRATT, V.R., *A practical decision method for propositional dynamic logic*, 10th ACM STOC, Proceedings 1978, pp.326-337.
- [PRA78b] PRATT, V.R., *A Near optimal method for reasoning about actions*, MIT/LCS/TM-113 (September 1978). (Also JCSS, Vol.20 (1980), pp.231-254.)
- [PRA79] PRATT, V.R., *Process logic*, 6th ACM SPPL, Proceedings 1979, pp.93-100.
- [PRA79a] PRATT, V.R., *Models of program logics*, 20th IEEE SFCS, Proceedings 1979, pp.115-122.
- [PRA79b] PRATT, V.R., *Dynamic algebras: examples, constructions, applications*, MIT/LCS/TM-138 (May 1979).
- [PRA79c] PRATT, V.R., *Six lectures on dynamic logic*, MIT/LCS/TM-117 (December 1979).
- [PRA80] PRATT, V.R., *Dynamic Algebras and the nature of induction*, MIT/LCS/TM-159 (March 1980). (A version of this paper appeared in 12th ACM STOC, Proceedings 1980, pp.22-28.).
- [PRA80b] PRATT, V.R., *Application of modal logic to programming*, Studia Logica, Vol.39, 2/3 (1980) pp.257-274.
- [PRA81] PRATT, V.R., *A decidable mu-calculus: preliminary report*, 22th IEEE SFCS, Proceedings 1981, pp.421-427.
- [PRA81b] PRATT, V.R., *Dynamic Logic, Logic, Methodology and Philosophy of Science VI*, ed. L.J.Cohen et Al., (1981) North-Holland.
- [PRA81c] PRATT, V.R., *Using graphs to understand PDL*, LP proc. 1981, ed. Dexter Kozen, LNCS 131, pp.387-396.
- [RAD84] RADEV, Slavian, *Extension of PDL and consequence relations* 5th SCT, Proceedings 1984, LNCS 208, pp.251-264.
- [SEG71] SEGERBERG, K., *An Essay in Classical Modal Logic*, 3 Volumes,

University of Uppsala (1971).

- [SEG77] SEGERBERG, K., *A completeness theorem in the modal logic of of programs*, preliminary report, *Notices of the AMS*, Vol.24, 6, A-552 (October 1977).
- [SEG78] SEGERBERG, K., *A completeness theorem in the modal logic of of programs*, *Universal Algebra and Applications, Papers presented at Stefan Banach International Mathematical Center at the semester "Universal Algebra and Applications" held February 15 - June 9, 1978*, ed. Tadeuz Traczyk, Banach Center Publications, Vol.9, PWN - Polish Scientific Publishers, Warsaw 1982, pp.31-46.
- [SEG80] SEGERBERG, K., *Applying modal logic*, *Studia Logica*, Vol.39, 2/3 (1980) pp.275-295.
- [SEG82] SEGERBERG, K., *The logic of deliberate action*, *JPL*, Vol.11, No.2 (May 1982) pp.233-254.
- [SEG85] SEGERBERG, K., *Models for actions*, *Analytical Philosophy in Comparative Perspective*, ed. B.K. Matilal and J.L. Shaw, D.Reidel, Dordrecht, Holland (1985) pp.161-171.
- [STR77] STROY, J., *Denotational Semantics: The Scott-Strachey approach to programming language theory*, MIT Press, Cambridge, Mass. (1977).
- [STR81] STREETT, R.S., *Propositional dynamic logic of looping and converse*, 13th ACM STOC, *Proceedings 1981*, pp.375-383.
- [STR82] STREETT, R.S., *Propositional dynamic logic of looping and converse is elementarily decidable*, *I&C*, Vol.54, (1982) pp.121-141.
- [T&M85] TIOMKIN, M.L. & MAKOWSKY, J.A., *Propositional dynamic logic with local assignments*, *TCS*, Vol.36 (1985), pp.71-87.
- [T&V84] TINCHEV, Tinko & VAKARELOV, Dimiter, *Propositional dynamic logics with counters and stacks* 5th SCT, *Proceedings 1984*, LNCS 208, pp.364-374.
- [TIU82] TIURYN, Jerzy, Review of [HAR79], *JSL*, Vol.47, No.2, (June, 1982) pp.453-454.
- [VAK80] VAKARELOV, D., *Filtration theorem for Dynamic Algebras with tests and inverse operator*, *Logics of Programs and their Applications, Proceedings 1980*, ed. A. Salwicki, LNCS 148, pp.314-324.
- [VAR85] VARDI, M., *The Taming of Converse: Reasoning about Two-way Computations*, LP proc, 1985, ed. Rohit Parikh, LNCS 193, pp.413-424.

- [VB84] VAN BENTHEM, Johan, *Correspondence Theory* Chapter II.4 of *Handbook of Philosophical Logic*, Vol.II, D. Reidel, Dordrecht, Holland (1984) pp.167-247.
- [WAN85] WAND, Mitchell, Review of [GOL82], JSL, Vol.50, No.3 (September 1985) pp.854-855.
- [X81] *Transcript of Panel Discussion*, LP proc. 1981, ed. Dexter Kozen, LNCS 131, pp.406-429.

END

1|0.0|8.88

FIN