

SYSTEM SPECIFICATION: A DEONTIC APPROACH

Samit Khosla

July 1988

Department of Computing
Imperial College of Science and Technology
University of London
London SW7 2BZ
United Kingdom

Submitted to the University of London in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy

ABSTRACT

This dissertation proposes a formal framework for the specification of dynamic multi-component computer based systems such as distributed databases and electronic mailing networks.

The framework itself is presented as a number of formal logics which, from an initial logic, are all ordered (conservative) extensions of the initial one and are used to formally model the concepts we have chosen. The idea is that each extension deals with one or more new concepts, whilst preserving and perhaps making use of the manner in which previous concepts are modelled.

As the basis of our framework we propose and motivate the idea of a system being modelled in terms of a collection of global, perhaps incomplete, information states called scenarios and a notion of action for describing movement between these scenarios. We employ a novel modal extension of many sorted first order logic for this. In order to model the behaviour of a system we extend this by introducing the deontic concepts of permissible and obligatory action (and subsequently property) this then allows us to state when actions may and must happen as opposed to just describing the effects of such actions. Two further perhaps less central extensions are also incorporated: action combinators and agents. The former is used to describe more complex actions such as non deterministically and parallel composed actions and the later is used to associate actions with the things (agents) which perform them. The introduction of action combinators and agents also serves to illustrate the advantages and benefits of separating behaviour from action description.

The major issue which is discussed is the discrimination and relationship between the way in which a systems behaviour is described and the way in which mechanisms that allow a system to "change" (and hence make possible desired behaviour) are modelled. A clean and smooth formal integration of these two related concepts is the major contribution of this dissertation.

Acknowledgements

It is my pleasure to take this opportunity to thank all those who have helped me to produce this work.

I am especially grateful to Tom Maibaum, my supervisor, for all the guidance, insight and encouragement he has provided. Tom has not only been a supervisor but also a friend, providing opportunity and help without being asked. Special thanks are also due to Martin Sadler whose views on logic have undoubtedly affected my own.

I would also like to express my thanks to my fellow students and researches at Imperial. Because of their number I shall simply refer to them as the fourth and fifth floors of the department of computing, Imperial College. I am in all their debts for both the work and social environment they have provided me. I am also grateful to Paulo Veloso, a friend and visiting Professor from Brazil, for the many technical discussions we have had about our work.

Finally I wish to thank my family, Mum, Dad, Ashok, Namrata and Maneet, for being so supportive and understanding, without which none of this would be possible.

I dedicate this work to my parents, Kanta and Chaman Lal Khosla.

Jai Matha Kī Jai.

Table of Contents

1 . Introduction	6
1.1. Historical Perspective	7
1.2. Application Domain	14
1.3. The Framework	16
1.4. Semantic Models	21
1.5. Methodology	23
1.6. Overview	27
2 . Scenarios and Actions	30
2.1. Introduction	31
2.2. SFOAL (<u>S</u> orted <u>F</u> irst <u>O</u> rders <u>A</u> ction <u>L</u> ogic)	37
2.3. SFOAL Specifications	43
2.4. SFOAL Models	61
2.5. An Example	70
2.6. Conclusions	84
3 . Deontic Logic and Behaviour	87
3.1. Introduction	88
3.2. DAL (<u>D</u> eontic <u>A</u> ction <u>L</u> ogic)	98
3.3. DAL Specifications	104
3.4. An Example	122
3.5. Conclusions	137

4 . Action Combinators	139
4.1. Introduction	140
4.2. Primary Action Combinators	146
4.2.1. Structural Aspects	147
4.2.2. Normative Aspects	161
4.3. Secondary Action Combinators	174
4.4. CDAL (<u>C</u> ombinatory <u>D</u> eontic <u>A</u> ction <u>L</u> ogic)	186
4.5. An Example	190
 5 . Agents	 202
5.1. Introduction	203
5.2. DL (<u>D</u> eontic <u>L</u> ogic)	207
5.3. DL Specifications	216
5.4. An Example	227
 6 . Conclusions	 240
6.1. Introduction	241
6.2. Potential Extensions	245
6.3. Exploitation	250
6.4. Research Collaboration	254
 Bibliography	 256

CHAPTER ONE

INTRODUCTION

1.1 Historical Perspective.

As the title of this dissertation suggests, our work is concerned with a specific approach to the problem of system specification. The product of our work is a formal framework for the specification of multi-component computer based systems such as distributed databases and electronic mailing networks. Hence, the accent is on "total" system specification rather than on software alone. Indeed, one of our central motivations has been the creation of a formalism for stating the requirements of such systems as opposed to their design. In short, our aim has been to create the kernel of a specification "approach" without getting enmeshed in the details of producing a specification language.

The key feature of our "approach" is an explicit ability to distinguish between the description and prescription of a systems dynamics. A common mistake in most specification formalisms is the assumption that describing change implicitly defines behaviour; even when the distinction between these two notions is desirable no attempt is made to provide a smooth integration between them. This can be seen in specification formalisms such as VDM [Jo80] and Z [Su85] in which the pre and post - condition style (where actions are described by properties holding before and after the action is performed) of operation description is used. Here the behaviour of a system is never explicitly given, instead it is tacitly assumed that operations are able to occur when their pre - conditions are satisfied. We will demonstrate that this not only causes unnecessary complexity in the description of operations but also leads to confusion in reading and reasoning about a system's dynamics.

In contrast to this we shall propose a two tiered structure designed to separate the description of operations (which from now onwards we refer to as actions) and the behaviour of a system. The former is achieved by a novel variant on the traditional pre and post condition style description of actions. In order to model behaviour, we introduce the deontic concepts of permissible and obligatory action (and, subsequently, property). This then allows us to state when actions may and must happen as opposed to just describing the effects of such actions.

In order to understand the route we have followed, it may be instructive to place it in a historical perspective. As we shall not have room to cover this vast topic in its entirety, we will focus our efforts upon the motivating issues behind our work. Perhaps our

major decision has been whether or not to opt for an algebraic or imperative (state based) foundation. Although the strong algebraic tradition could not be ignored, we felt that in order to make precise a distinction between action description and system behaviour an imperative approach was essential. Below we will provide glimpses of both traditions. We start by summarising the emergence of the "specification" field itself.

The last one and a half decades or so has shown that whatever the role computers and computer related processes play in society they are constantly expanding their domain of application. The image of the computer as a high speed (meticulous) clerk has been replaced with a more intelligent exterior. They are now employed in almost every aspect of daily life; from distributed booking and information systems, such as airline reservation and control applications, to vendors of "expertise", such as first defence medical diagnostic aids. One consequence of this rapid expansion has been the inability of the empirically based methods and techniques of the traditional "systems analyst" to cope with the diversity, size and sheer complexity of these new applications.

The argument that contemporary computer systems need to be constructed on a more formal footing is now a well accepted one. There are many reasons for this conviction almost all of which have their origins in software development. Claims such as "the impossibility of writing 'correct' programs" began to ring true in the ears of even the most experienced systems designers. Not surprisingly then, a large amount of research was initiated and directed at producing "correct" code.

The pioneering work of Floyd [Flo67] and Hoare [Ho69] proposed formal logical apparatus by means of which programs or program fragments could be analysed. Briefly, the strategy was to assert a logical expression representing the state of affairs before a program's execution and another describing the desired state of affairs on program termination. Given these two assertions - which became known as the input/output relation of a program - the aim was to prove that the program changed the state of affairs expressed by the input expression into a state of affairs expressed by the output expression. This was done by providing rules for how logical expressions changed as a result of executing primitive program statements.

Although the main intent of these early formalisms had to do with the validation of programs, perhaps their more significant contribution lay in the way that they highlighted and hinted at how programs should be constructed. The realisation that formal languages and their associated semantics could not only help to check extant

programs but also to design better software was a major turning point in the history of computer science. It was at this stage that the phrase 'formal specification' took root. A precise taxonomy of this development is neither possible nor necessary for our present purposes. What is relevant is the birth of a new perspective - the use of formal mathematical models in the design of computer software.

This perspective was accompanied by a new paradigm of high level programming languages whose core philosophy was based on a "what" rather than "how" emphasis. The basic tenet of this philosophy is the observation that most software errors arise in the process of encoding the solution to a problem or task. One way to avoid this is to strive for programming environments in which it is only necessary to 'state' the problem, leaving it implicit as to how the solution is arrived at. This dream of automatic programming was pursued with much rigour but it was quickly realised that the 'dream' had a non trivial realisation, if any. One benefit of this activity was the realisation that informal statements of problems were not an appropriate basis for such programming systems. Problem representations needed to be precise in order to arrive at recognisably satisfactory solutions.

The shift from an imperative to a more declarative perspective was not confined to programming languages. The recognition that data structures have abstract properties and that these are helpful if not crucial in the characterisation of the programs that use them - again, this pioneering observation is due to Hoare [Ho72A] - led to the development of the ADT (Abstract Data Type) notion. Starting with such notions as ADT (see for instance [GH78] and [Gu77]), data structures began to play a prominent role in the design of computer systems. Once Zilles [Zi73] had made the proposal that an ADT can be thought of as an algebra, i.e. a set of objects together with a collection of operations on them, the domain of application of ADTs expanded. The algebraic theories of [ADJ77] and [Eh82] saw the use of ADTs in fields such as database design [DMW82] and programming [BG80].

The success of algebraic ADTs, both in their application and diversity, is fundamentally due to their capacity to distinguish between what a program does and how it does it. It is from this very distinction that the term "specification" arises. Thus a specification is a statement of want. Its co-term "implementation" had the early use of some program or suite of programs which could be executed in order to achieve the intent expressed by a specification. Later on we shall see how this relationship may be broadened.

The acceptance of ADT's as units of specification gave rise to a number of developments most of which grew out of the associated algebraic theories that provided their foundation. Two important spinoffs were the emergence of term rewriting systems such as Hope [BMS80] and the attention given to structuring specifications.

The latter has received immense attention; starting with the pioneering paper by Liskov and Zilles [LZ75] and still continuing to date, see for instance [BG80],[SW82] and [W83]. The basic idea was simple and was itself based on the lessons learned from programming languages: in order to construct large specifications, mechanisms for 'putting together' smaller specifications were needed. Just as in programming languages the notions of 'reusability' (often associated with parametrisation) and 'modularity' were employed.

The former originated as an attempt to realise the dream (we mentioned earlier) of 'automatic' programming. This has since been diluted to a lesser claim of facilitating executable specifications. The only problem with this is that the 'specifications' are constrained by the limiting (equational) expressive powers of such languages. Again attempts to overcome this provided much food for thought; proposals concerning the transformation of non executable specifications into executable ones brought with it the more general idea of refinement - which was also something discussed as a structuring concept. Briefly, refinement is the process by which one specification is transformed into another such that the latter displays more details. We shall return to this idea later.

An important disadvantage of languages such as Hope [BMS80] (and hence also the algebraic foundations on which they are based) is their unacceptability to the large majority of computer practitioners whose programming paradigm remains a state based (imperative) one. The need to follow a constructive approach to data structures, for example, brings with it an unacceptable expense of processing; which will probably remain until novel, currently prototype, hardware architectures establish themselves. For this and other reasons a parallel development of specification languages and associated foundations has been taking place.

State based, also referred to as imperative, specification approaches reject the algebraic paradigm for many reasons. Perhaps their major dissatisfaction lies in the way algebraist model the idea of an action; for algebraists there is no way to formally distinguish between a functional relationship and an action. This is perhaps a little unfair as within a specification it may be possible to distinguish between these two by the specific types of

objects involved. However, the problem becomes more acute if we attempt to distinguish between an action's description and its prescription (when it is to be performed). If we are to discriminate between the description and prescription of actions it is essential to distinguish actions from ordinary functions; imagine trying to prescribe the behaviour of the successor function over integers. We believe that a clear distinction between static functional relationships and actions should be made in order to avoid this kind of confusion.

In contrast, to the algebraic view, the availability of a formal 'state' object, which describes the system's state of affairs, can be used to distinguish these two rather different notions. Actions are seen as operations on states, transforming one state into another and, hence, one state of affairs into another, whereas functions are reserved to model functional relationships and properties within a state.

The main disadvantage of this view is the difficulty in formally constraining actions so that they do not have effects, referred to as side effects, which are not intended. Unfortunately, this is a problem which is inherent in the approach. Actions are described in a before and after style: a statement indicative of the state of affairs before the action is performed and a statement which is indicative of the state of affairs after the action has been performed. The problem is that these statements are only indicative about the states of affairs they represent: they only represent *an* identified property of that state. Nothing is said about what happens to other properties of these states. This problem has come to be known as "the frame problem".

Various proposals have been made as to the solution of this problem, see for instance [Ko79] or [Sh86]: most seem to rely on some sort of meta constraint(s) which stipulate an idea of minimal change. In other words only that which is mentioned in the description of the action changes and all else remains as before. For all but the simplest of action descriptions - concerned, for example, with purely monotonic actions (actions which never refute any known property or fact) - this is inadequate. There may for example be situations in which, in order to achieve the desired effect of an action, it is necessary to choose between a number of possible (equally "minimal") changes. Thus, the simple minded meta constraint approach fails. Nevertheless we, as have others, shall defend the state based paradigm as both natural and manageable.

There are many ways in which to formulate the concept of states; it may for example be a very explicit object as found in programming languages such as Planner [He69] and

Smalltalk80 [Small]. Interestingly enough later developments in the algebraic school, [Ma81] and [CCF81] for instance, incorporated explicit state objects into their formalisms in order to distinguish between actions and functional relationships more easily. Actions were modelled as functions which had as one of their arguments a state object.

Another approach was to leave the idea of state implicit; relegating it from the syntactic level to the semantic. In time this has gained favour over the explicit inclusion of states, due mainly to the clumsiness and verbosity of the latter. This is not hard to appreciate given that the explicit approach gave no more guarantee of avoiding side effects and in some cases constrained a specification unnecessarily by demanding the description of an initial state. Amongst the advocates for implicit states are the VDM [Jo80] school as well as many modal logic based formalisms, see for instance [CCF81], [MSF80] and [GMS83]. Modal and temporal logics are particularly suited to this task as their semantics are based upon the idea of possible worlds and accessibility structures between these worlds. There is then an immediate analogy between possible worlds and states and accessibility relations and state change. Indeed it is the interpretation and characterisation of the latter which distinguishes between the various modal and temporal systems.

Much of our work can be identified as belonging to the modal logic (implicit state) approach; there are many reasons for us adopting this line but perhaps the most basic is our dislike and dissatisfaction with the manner in which actions are characterised in the algebraic school. We sympathise with the view that state changing functions should be distinguished clearly from state based functional relationships: this point will be made more forcefully and also expanded upon in the next chapter. This is even more crucial if we are to draw the further distinction between the description and prescription of an action. Further, as we have already indicated the explicit state approach is not in general abstract enough to circumvent the design of an explicit state entity and also the values of an initial state. In addition it makes specifications more cluttered and necessitates the careful manipulation of a global parameter, for no discernible benefit over the implicit approach. Our own belief is that the explicit state approach is more useful at the *design* stage of system development.

Although we have given an overview of the two major schools in specification theory, there are many specification languages which do not completely fall into either school. The specification language Z [Su84] is a good example of this: Z takes as its foundations

the Zermelo-Frankel foundations of set theory and from this builds a mixture of algebraic and logical machinery. In its most basic form Z is a specification language based on the construction of sets that has been likened by at least one colleague to a set based type theory with prominent constructive overtones [Vi86]. However, it has also been said that Z does not seem to possess the clean cut foundations of constructive type theory as found in [M-L79], for example. It is not within the scope of our present aims to discuss the entire philosophy of specification languages like Z and we will restrict ourselves to the manner in which they deal with modelling system dynamics. In this respect Z can be seen to fall into the VDM class of an implicit state based approach.

Having established an extremely brief and partial history of the development of specification theory we shall devote the remainder of this chapter to providing a fuller picture of our approach - comparing it where necessary to other work. Before we do this, perhaps we should point out that a direct comparison between our work and any of the plethora of specification languages that exist is not possible. We are not in the business of designing a specification language. We have not for instance defined any "built in" types or data structures, nor have we suggested any structuring mechanisms. Having said this we do believe that our framework is general enough to accommodate such features and will comment, whenever appropriate, on how this may be done.

1.2 Application Domain.

The first thing to establish in the design of any specification language or, as in our case, specification approach is a target domain of application. As we have already seen most contemporary specification languages owe much of their origins to the software production life cycle; these early beginnings have more recently been enveloped within a more expansive "total system" setting. This usually leads to the modelling of objects and processes not designated to be given representations in a computer; the simple activity of alerting a human operator to shut down some device is a good example of both. The human serves as an example of a non software based process and the device he is instructed to shut down as an example of an entity which may be doing the instructing.

This is not to say that specification languages have turned into all encompassing monolithic formalisms; there are numerous examples of specification languages which embody more than one kind of formal representation mechanism - perhaps to model different aspects of a system. In the context of database specification, for example, there have been some attempts, [SK86] for instance, to model updates in the usual pre and post condition fashion and then model the behaviour of a system using a Petri net [Pe79] variant. Quite apart from such mixing of formalisms there also exist quite specialist specification languages such as CCS [Mi80] and CSP [Ho83] which deal with describing concurrent behaviour.

As far as our own work is concerned we have centred upon providing a uniform integral kernel from which one or more requirement specification languages can be built. By requirement specification languages we mean languages designed to express the required characteristics of a system - as opposed to those used to express particular designs of a system. The difference between these is often quite subtle; one way in which they may be distinguished relies on judging them according to purpose. A requirement specification language can be judged by how well it allows us to state all the characteristics and properties required of some system. A design specification language can be judged by how well it allows one to encode these requirements into a chosen design biased formalism (corresponding to the particular design approach adopted). Of course, the line between these will always remain vague since any formalism employed to express requirements will possess some level of design bias - our framework for instance features the idea of a scenario or state of information.

It follows then, that a requirement specification language, and even more so the kernel of one, should, by design, (no pun intended) provide an adequate but minimal formalism. Here we use minimal in the sense of not building in (*a priori*) too many constraining design features. It is with this in mind that we have omitted augmenting our framework with standard data constructs such as those of sets and sequences. Our feeling is that such machinery should be "borrowed" from existing literature and incorporated as specification specific components or dealt with when extending our framework to include more specification language structure. The successful use of our framework in the description of traditional data structures such as stacks and queues, see for example [KMS85] and [Go87] respectively, lends a good deal of credence to this belief.

This last point also serves to highlight the fact that the framework need not exclusively be considered a requirements specification vehicle but also as a design specification medium. Again this immediately follows from what we were saying earlier; design has some bias towards how a system is constructed whereas requirements only capture the desired properties of a system. Given this together with the fact that specific data structures can be encoded it is not difficult to see how system designs can be expressed; essentially by refining a system's requirements to incorporate design bias data structures. Precursors and variants of our framework have been used to design network protocols [Li86] and lift systems [FP87]. The idea of refinement is discussed more fully below when we talk about methodologies.

1.3 The Framework.

As the basis of our framework we propose and motivate the idea of a system being modelled in terms of a collection of global, and in general, incomplete, information states called scenarios. Here each scenario represents the body of information we have about a system's state of affairs at any given moment. We shall delay the question of why we chose to model global and (in general) incomplete states of information and deal first with the kinds of information specifications will incorporate.

Broadly speaking we think of a specification as consisting of three types of information: static information, action descriptions and action prescriptions. The first two of these combine to give a transition system style of specification, whereas the last acts as a constraint on transitions, in order to define a desired behaviour.

By the phrase *static information* we mean that part of a specification which models the entities, properties and relationships which exist in a system. In order to represent static information we use a sorted classical first order logic as may be found in [En72] for instance. For our present purposes, the reader may like to think of a *sorted* first order logic as being augmented with a simple typing structure. Instead of a homogeneous universe of discourse, we deal with one which is pre-partitioned into a number of sorts/types, each sort representing a kind of object. An action, for instance, would belong to an action sort, whereas a person would belong to the sort representing people.

The more standard machinery of first order logic may then be used to model the system in hand in the usual manner. Predicates are used to signify properties about and relationships between (sorted) objects, functions signify functional relationships between such objects and the logical connectives and quantifiers are used in the traditional way in order to characterise more complex system statements. The only restriction is that all this machinery must respect the sort or type of the objects with which it deals - we shall see what this entails in more detail in the next chapter. This provides our framework with a very loose - yet quite expressive - foundation; most of the data structuring found in VDM [Jo86], for example, can be constructed easily in a sorted first order setting. For instance, if given the appropriate construction functions, it is not difficult to characterise sets and sequences of particular sorts of objects. In chapter three we shall see how we can describe sequences of actions in this manner.

The second kind of information is action description and we must be clear at the outset to distinguish actions from action descriptions. For us, actions are the instruments of movement between states of affairs, whereas action descriptions are indicative of the information we have of such state changes. In other words, action descriptions indicate the information we have about the change just as scenarios represent the information we have about states of affairs. This is an important point and will be exploited to some advantage in the next chapter. As we have already said, we adopt a pre/post condition style of action description; the pre-condition is indicative of the scenario before the action is performed whereas, the post-condition is indicative of the scenario resulting from performing the action.

We have chosen to follow Goldblatt [Gol82] and introduce a modal connective in order to represent action descriptions. Although this creates the extra burden of having to relate this new connective to the traditional first order connectives, it does provide us with certain advantages over other approaches. The two main benefits are: being able to state action descriptions as pre-conditions and post-conditions and being able to reason about properties of actions (such as sequencing) without resorting to meta level rules (as is required in some specification languages of the VDM school). We shall see examples of action descriptions as pre and post conditions of other action descriptions in the next chapter and examples of sequencing rules in chapter four. Other benefits of our approach are that we explicitly attach a *doer* to our notion of actions, and hence to action descriptions, and we also allow the construction of more complex action descriptions from more primitive ones by introducing action combinators.

Another difference between our approach and more traditional ways of describing actions can be found on closer inspection of the role pre-conditions play. In most specification languages a pre-condition is used both to identify the context within which an action is performed and/or an implicit approval of that action being allowed to be performed. As far as we are concerned pre-conditions used in action descriptions should only be concerned with the former.

We take the view that the presence of a pre-condition (perhaps a better word would be pre-context) stands only to identify the context in which an action is performed. This lends itself well to the idea that a particular action α can have different results (post-conditions) depending upon the context in which it is performed. The absence of a pre-condition is taken to mean that the result of executing the action in question is not context dependent, not the traditional intuition that the action is allowed to be performed.

Last, but certainly not least, we have information about action prescriptions; this is concerned with the other half of the pre-condition intuition. Action prescriptions are meant to convey when actions may or must occur, via the deontic concepts of permissible and obligatory action. This is perhaps the most original component of our framework; it involves the partitioning of all scenarios into two kinds, normative and non normative. Intuitively speaking a normative scenario is one which is behaviourally desirable and has been deontically sanctioned, whereas a non normative scenario is behaviourally undesirable and has been deontically prohibited. The idea is simple: all actions which lead to normative scenarios are defined as permissible actions and all permissible actions lead to normative scenarios. The converse holds for forbidden/prohibited actions: a prohibited action is not permitted and leads to a non normative scenario and all actions which result in non normative scenarios are prohibited.

Thus the definition of the permissible action operator is easy to give - it is equivalent to a special action description which states that performing that action leads to a normative scenario. This definitional characterisation of permissions is further restricted to apply only in the context of normative scenarios. Hence, a permissible action is only guaranteed to lead to a normative scenario if it is performed from a normative scenario; it is left undecided as to whether this will be the case if it is performed from a non normative scenario. A specification may force the decision either way; this feature is exploited under the banner of fault tolerance. It may be, for instance, that some non permitted actions are tolerated (expected in an operational system context) - leading to non normative scenarios; other actions may then be explicitly described as leading back to the normative domain. This method of explicitly (not employing deontic operators) describing actions as leading to non normative scenarios is explored more fully in chapter three.

Obligations, or more strictly obligatory acts, are partially characterised in terms of permissible acts. Intuitively, we may explain it thus: "If an action α is obliged then it is also permitted and further, no action other than α is permitted". This has the consequence of forcing a very strong interpretation on the obligation operator; in effect, when an action is obliged the next action performed must be the obligatory one. Although this rather strong interpretation is presented first, we do weaken it so that in a multi agent context one agent may perform an action that derogates another agent's obligation. This does however still mean that a specific agent is still restricted to one action if it is obligatory; although we do not go into great detail we also show how this

may be weakened so that one action from a particular collection of actions is obligatory. Hence, a set of actions may be said to be obligatory, with the effect that one action in that set must be the next action performed.

This characterisation of obligations in terms of permission is however only part of the intuition of obligatory acts; the remainder is supplied by a more dynamic (transition based) intuition of obligations. Briefly, we think it natural (taking a lead from its use in English - and we suspect other languages) that obligations be thought of as an inherently transitory phenomenon. When an obligation to do something exists, the default is to treat it as though after the obliged action has been performed the obligation for it to occur again is lost. If this were not the case, obligations would, once incurred, persist forever. This is not to say that an obliged action cannot cause the same action to be obliged again, merely that the default should not force this. Hence, an obligatory act carries with it its own implicit derogation. In contrast to this, permissions are more intuitively thought of as persisting until they are explicitly revoked. We can see this distinction more clearly with the aid of an example. Suppose that the manager of an employee grants permission for a subordinate to use the company's new swish management dining room (a privilege granted for outstanding achievement, say); in this case it is natural to assume that he may use this facility until it is explicitly revoked. In comparison, assume that the same subordinate is now obliged to use the managerial dining room (as his services are required to impress an important client); in this case, our intuitive default tells us that once the subordinate has attended this special lunch, he will have discharged his obligation and so will not be *forced* to use that particular dining room again. Once again, we stress the importance of the word default. It may, for example, be that the person in question incurs a similar obligation as the result of the first - he may for instance be obliged by the important client to meet again for lunch the next day.

This transitional aspect of obligations has been one of the most difficult to capture, essentially due to the problem it creates for the persistency of more stable deontic information which prohibits or permits various actions. To see this, imagine a scenario in which a number of actions are permitted and a number of actions are forbidden and refer to this as the p-structure of that particular scenario. Further, assume that one of the permitted actions is performed and leads to a scenario in which some action is obligatory. This means that the resulting scenario is one in which all actions other than the obligatory one are not permitted; but now suppose that the obligation incurred is discharged. What is the p-structure of the resulting scenario? According to our previously explained intuition it should be the p-structure that obtained before the

obligation was incurred. So far, we have only been able to supply a semantic characterisation for this transitional aspect of obligations. This is hardly surprising when, as we shall see later, this semantics is expressed in terms of transition constraints and incorporates the same problems as trying to axiomatise the frame rule/assumption. To summarise the transitional aspects of obligations may be thought of as temporarily hiding the current p-structure and restoring it once the obligation has been discharged. This is reflected in a number of model level constraints, which are extended to deal with obligations in the context of multiple agents and sequences of obligations (a number of obligatory acts occurring one after another in some specified order). It is to this notion of models to which we now turn.

1.4 Semantic Models.

The traditional role of a model in specification languages has been to provide a semantical basis from which to: prove the validity of certain expressions, prove the invalidity of seemingly intuitive expressions by providing counterexamples, and help in the proof of certain expressions by providing a more intuitively appealing (and usually more mathematically manipulable) formalism. Models are constructed by providing an interpretation mapping from the specification formalism to some other formalism, such that the language constructs and operators in the former are assigned corresponding denotations in the latter. It is then common to show that all expressions which can be shown to hold in some specification formalism also hold in the chosen model formalism and vice versa. The former is known as soundness and the latter completeness.

We have chosen not to adopt this traditional aspect of underpinning our logic by some other language and proving soundness and completeness. Although this logicist perspective is no doubt an important one we have chosen to focus our efforts on more specification orientated issues. Those readers interested in the former viewpoint, in relation to models of our framework, are referred to [CC87] wherein the authors provide soundness and partial completeness results for significant fragments of our framework. In this work we shall look upon models as a way of explicitly characterising the scenario transition system which underlies our whole approach.

In brief, our characterisation of models consists of a collection of scenarios representing the states of information about a system and a collection of scenario transforming (total) functions representing transitions between such states of information. Each scenario is defined to be a theory extension of the specification in question, whereas each function corresponds to one action in the specification. The nature (extension) of these functions is taken from the description of the actions to which they correspond. Thus, if an action α was described, in some scenario s , as leading to a scenario in which some property ϕ held, then the function corresponding to α would when applied to the scenario s lead to some other scenario s' in which the property ϕ did indeed hold. This rather weak correspondence between actions and the functions which are used to model them is augmented with an implicit frame axiom in order that the resultant scenario is not wildly different from the scenario in which the action was performed. This informal meta device is necessary because action descriptions are only indicative of scenarios which result from performing actions; that is, they only specify the desired result. Without the

frame axiom (which has the effect of preserving all those properties an action does not effect) it would be quite conceivable for actions to transform one scenario into another without the latter bearing any relation to the former except for exhibiting the property indicated in the action's post condition.

In addition to the laws governing how functions model actions, there are also a number of transition constraints which govern the transitional aspects of obligatory actions. These can best be seen as explicit *frame* type conditions, which are used to reinstate restructures (the persistent information regarding the permissibility of actions in a scenario) after the discharge of an obligation (or sequence thereof). The nature of these constraints makes it impossible for a completeness result for our framework, indeed we have not found an acceptable way to reflect them in our logical apparatus. We shall see the details of these constraints in chapter three. For the moment they may best be thought of as providing a complementary (at a model rather than at a specification level) perspective on the construction of dynamic systems.

In contrast to other state transition models we have decided to adopt the notion of underdeterminedness rather than nondeterminedness in order to model *looseness* of specifications. This is reflected directly in our decision to characterise transitions by functions rather than a relationship between scenarios. This in turn was based on the intuition that the former seems to correspond more naturally to the relationship between a specification and potential implementations. In general a specification has many implementations just as in our approach a specification has many models. However, we do not therefore assume that because a specification does not completely determine a particular action that the corresponding implementation will be nondeterministic (which corresponds to the relational setting of providing more than one resultant scenario for an action) but rather that there are perhaps many different acceptable scenario transforming functions.

Hence, a specification which does not completely determine an action will in general have a number of models each of which will have a different scenario transforming function corresponding to the different possible outcomes of performing that action. Unfortunately, this approach also makes it impossible to describe truly nondeterministic actions. However, it is not difficult to extend our framework to a relational setting, this having in fact already been done in [CC87] wherein the authors provides a completeness result for the action description fragment of our framework. The reason for not doing so here is to highlight the role of underdetermined information in transition systems.

1.5 Methodology.

In this section we would like to present the overall picture within which our framework sits; we shall do this rather informally with the aim of giving the reader a feel for our approach - as well as some justification for it. Some of the more formal concepts upon which these ideas are built have already been mentioned above. Because our primary aim had been the development of a requirements specification formalism, we needed to be acutely aware of a refinement methodology, something which could aid the smooth transition from requirements through design and on to implementation.

Although our framework does not incorporate an explicit methodology it has been developed against an implicit assumption of a top-down global refinement strategy. Briefly, this entails a layered approach to system construction, where, starting from a system's requirements and passing through the system's design one arrives at an implementation which is executable and in harmony with the system's requirements. There may be many layers of refinement in a system's construction, each revealing more detail and implementation bias. Here the distinction between design and implementation is analogous to the difference between requirements and design; an implementation adds implementation bias to a design; for example, we may implement information as Pascal [Wir71] records even though the corresponding design does not dictate this choice.

This process of refinement is supported by the notion of logical implementation the details of which may be found in [MVS85]. The basic idea which underlies this notion is an interpretation between logical theories; the language and axioms of one logical theory are translated into the language of another logical theory such that the properties of the first are preserved in the second. Given that the theory into which translation is made represents a more biased and less abstract collection of concepts and a translation is possible it will in general be possible to refine a more abstract specification by interperating it into such a theory. In [MVS85] the authors demonstrate this process by "logically implementing" the theory embodying the finite sets of natural numbers by a theory representing finite lists of natural numbers. An example of this refinement process between theories (specifications) written in a precursor to our framework can be found in [KMS85] where the authors show how the notion of stacks may be logically implemented by pointers and arrays.

It is beyond the scope of this work to explain this refinement process in any more detail;

however, it is an important process to note as it has influenced other choices in the design of our framework. Amongst these are the decision to opt for a global state perspective and a proof theoretic account of incomplete information.

The reason for opting for global states of information can best be explained by observing the alternative route. The specification of systems each with their own state space, as found in the schema concept of Z [Mo84], does at first seem like an obvious structuring device. The reality however is quite different; in practice it is often difficult to relate system wide components and so express conditions which refer to properties in more than one state space. As an example of this, consider a communications protocol in which in order for two nodes of a network to communicate various properties in each node must hold. If we had started off by specifying the two nodes (each with its own local state space) in what way would we then characterise when communication could proceed? Clearly we would need some way of referring to states in each node, but it is not clear to us how this should best be done. It is almost certain that either some form of message passing metaphor would be required to establish a connection between local states or a treatment of concurrency based on interleaving or truly concurrent behaviour.

All these options seem unnecessary to us, there is a role for the specification (and hence communication between) related subsystems but it seems to us that one should start out with a global statement of requirements written in a formalism in which it is possible to refer to all system components without positing some synchronisation mechanism. This may then be refined so that an eventual implementation consists of a number of related subsystems. Although we shall not deal with it here, work is in hand to enable specifications written in our formalism to be refined into specifications which include CSP [Ho83] operators and allow such independent subsystems; the interested reader is referred to [HGQ87] for a more detailed treatment.

The idea of a refinement strategy has also influenced our treatment of incomplete information. In essence, our approach has been to view a specification as a theory (in some suitable logic) which models a specific system. Invariably such theories are incomplete, either due to the nature of the system involved or to our (present) understanding of it. An example of the former can be found in most real time systems which deal with uncertain environments. Consider, for instance, the specification of an underwater probe; it would be impossible to specify all the useful functions of such an instrument without knowing all the features of its intended environment. On the other hand it would be possible to incorporate more features as our understanding of the

environment grew (perhaps) from successive probing missions. This latter is an example of where an incomplete specification of a probe's objectives may be refined in light of more information.

Given that we have chosen to model systems as logical theories an incomplete specification will correspond to an incomplete theory. Thus a more complete specification (that is a refinement of the original) will lead to a more complete theory. There is nothing novel in this view and it is shared by many other specification approaches, VDM [Jo86] for example; what is perhaps slightly original is that our scenario (state) transition models of such specifications are also formulated in terms of incomplete theories. Hence instead of creating truth functional structures (which would need to be of at least three values, to allow for incompleteness) which represent states in some transition system semantics, we retain theories as denoting states of a transition system.

This has several advantages over the truth functional approaches, perhaps the most beneficial from our point of view is that it reinforces our view that a specification should be seen as defining an information structure *about* a system rather than defining an implementation *of* a system. We can see this if we note that a scenario (a state of information about a system) is a representative of a collection of states where all properties are either mapped to truth or to false. Hence the philosophy of modelling incomplete specifications as incomplete theories is retained when we deal with modelling incomplete states of such systems. Three valued truth systems have been introduced to model incomplete states in a truth functional semantics in [Jo86]; although such systems cannot be dismissed out of hand we feel that they are a more cumbersome alternative which unnecessarily fix an interpretation of *undefined* for gaps in a theory's information. Incomplete information cannot be equated (in general) to undefined information; we only need to consider the fact that this equality does not distinguish between unknowable information and that which is not *yet* known to see this. It is, we believe, better to leave such distinctions implicit in the nature of a system rather than trying to define in advance the denotation for every kind of knowledge gap. As a last point, it is also worth pointing out that scenario or state based reasoning also benefits from our approach; the notion of proof provides a conceptual way of *computing* the properties of a scenario. We shall see more about the role of incomplete information and how it is expressed in our framework in all of the following chapters.

We shall conclude this section by mentioning the work of some colleagues who have

created an explicit methodology, called SCS (Structured Common Sense), targeted specifically on constructing specifications in our framework. This work was carried out as part of an Alvey funded project called FOREST (standing for Formal Requirement Specification Techniques) which uses many of the ideas expressed in this dissertation as the basis for a formal specification logic called MAL (Modal Action Logic).

Their work is interesting because it uses a number of industry tried and accepted techniques as intermediary steps at arriving at formal specifications in MAL. The use of Entity - Relationship modelling, for example, is used to elicit the static structure of a system and this is subsequently transformed into expressions of the many sorted first order logic component of MAL. Data Flow Diagrams are used to model the information flow of a system with the aim of identifying the agents and actions of a system. Finally, agent event matrices are used to investigate causality relationships between agents, with a view to capturing the prescriptive descriptions of MAL. All these techniques are used to demonstrate how, starting from informal descriptions of requirements, formal statements in MAL can be reached. The techniques form the basis of a highly prescriptive method in which a collection of heuristics guides the practitioner from informal requirements through to a formal statement of requirements. The accent is very much on how to elicit and then arrive at formal requirements. The interested reader is referred to [FP87] wherein a full description of the method and MAL is given.

1.6 Overview.

This section maps out the remainder of this dissertation which is structured as follows. In the next chapter, chapter two, we shall see the core logic of our framework, this logic called SFOAL (Sorted First Orders Action Logic) is used to model what we shall refer to as the structural aspects of a system's requirements. The main components of this will be the description of entities, their properties and relationships, and the description of actions. In addition to providing an intuitive account of this logic using example expressions, we shall also provide a definition of models for specifications written in such a logic. This will take the form of a scenario transition system. We then introduce and explore a medium sized example, based around a small shipping company, expressing the its requirements of moving certain cargo by a number of ships between various ports. Having done this, we will then see how we can prove certain implicit properties of this requirement from both a specification and a scenario perspective. In the former we shall see what general properties the specification possesses and in the latter we shall see how augmenting the specification with additional information, specific to a scenario of the shipping system (a particular situation) can affect this reasoning process. We conclude the chapter by noting some deficiencies of SFOAL which motivates the deontic component of our framework.

Chapter three builds on the motivation introduced at the end of chapter two and introduces a conservative extension (one which preserves all the logical properties of the logic being extended) to the logic of chapter two to deal with the prescription of actions. The resulting logic is referred to as DAL (Deontic Action Logic). The two deontic notions of permission and obligation are tailored to allow us to express when an action *may* (is permitted) and when it *must* (is obliged to) occur. After an intuitive explanation for the rather novel axiomatisation of these concepts, the definition of models for specifications is altered to allow for this extension. This change reflects directly the transitional aspects of obligations mentioned above. The shipping company example of chapter two is then extended to incorporate prescriptive information which dictates how some aspects of the shipping company behave. This new logic is used to characterise and reason about which ships may sail between which ports and what and how much they are allowed to carry. Both the static and transitory aspects of permissible and obligatory actions are discussed in the context of this example. The example is also used to show how incomplete information, about which actions are permissible or obligatory, is to be viewed. We end the chapter by motivating the need for two new extensions to

DAL; the first is the need to be able to express more complex actions and the second to attach a doer (an agent or performer of an action) to actions. These are addressed in chapters four and five, respectively.

Chapter four is concerned with the formation of more complex actions from simpler ones. The logic of chapter three is extended to incorporate logical combinators, the resulting logic is called CDAL (Combinatory DAL). We have further divided these into primary and secondary combinators. Primary combinators are used to form more complex actions from more primitive ones. For example, the sequencing combinator takes two (ordered) actions and returns *an* action which when performed has the same effect of doing one action after another of the two original (more primitive) actions. The other combinators discussed are nondeterministic and parallel composition. All three combinators are discussed from both the descriptive and prescriptive aspects of our framework. Thus as well as characterising the effects of more complex actions in terms of simpler ones we also characterise how such actions are either permitted or obliged, given the deontic prescription of the simpler actions.

Secondary action combinators complement the action based deontic operators of chapter three by introducing a collection of operators which can be used to state when a particular *state of affairs* (a specific property) is either permitted or obliged. These operators can then be used in those situations where it is known which properties are permitted/obligatory, but it is not necessarily decided/known which action is permitted or obliged to bring about the desired property.

Both primary and secondary combinators are used in an example specification, written in CDAL, which describes the structure and behaviour of a small communication network.

Chapter five builds on the example of chapter four and motivates the idea of an agent by arguing for each node to have a certain amount of autonomy in the network. There are several advantages to this, apart from being a more faithful model of reality (a system is usually composed of a number of identifiable subsystems); it helps to structure specifications by localising (to agents) the way in which we prescribe the behaviour of a system. In addition it will allow us to be explicit about the interaction between various components of our system. For example, consider the situation where the effect of one agent performing an action results in an obligation for another agent.

Having introduced the rewriting of CDAL to incorporate agents we examine how this

change affects each component of our framework, starting with action descriptions, then action prescriptions and, finally, action combinators. The definition of models is also revised in order to reflect these changes. We end by discussing an example of a simple telephone exchange, highlighting the interactive aspects of agent behaviour. We have named the final logic DL (standing for Deontic Logic).

The final chapter (chapter six) summarises the main force of our work, looks at future work which may extend our framework, and how the framework has already been used.

CHAPTER TWO

SCENARIOS AND ACTIONS

2.1 Introduction.

In this chapter we shall present and discuss that part of our specification framework which deals with the *structural* requirements of systems; for us, this will involve creating the apparatus to describe the following:

- entities, the basic objects - conceptual/concrete, animate/inanimate - of a system, for example nodes in a local area network, or people in a banking system. As we shall soon see, a special type of entity is an action; actions are *names* for the observable units of change within a system. Examples of actions are updates to a database and the sending and receiving of messages in a distributed environment.
- properties of entities, these may be intrinsic to an entity or be of the kind which describe various relationships between entities. An example of the first kind would be the property stating whether or not a particular (network) node is "up" or not, an example of the second might be the fact that one action was a distinguishable component of another.
- action descriptions, which deal with characterising the effects of performing the actions we have identified, in other words representing the types of change which are named by actions.

There have been many proposals about how this should be achieved ranging from informal graphical techniques to rigorous mathematical formalisms. The case for a sound mathematical basis has been argued in many places - including the introduction to this thesis - and will not be presented again. However, there still remains some choice, broadly speaking; there are three major schools of thought on this: algebraic, logic based and set theoretic - although not all specification languages fit exactly into these divisions, VDM [Jo86] and Z [Su84] for example can be viewed as an amalgamation of at least two. Our approach will be of the logic based variety; there are numerous reasons for our choice ranging from the technical to the personal and we shall come across most of them in the remainder of this dissertation. There is, however, no technical difficulty in extending our work to include the set theoretic apparatus found in other approaches.

As the foundation for our framework we shall use a slight variant of classical first order

logic known as many sorted first order logic. This variant allows us to see the world as a collection of typed or sorted objects, one sort for each kind of object, rather than a single universe of objects, which is what standard first order logic requires. It seems natural to adopt a typed view of the world since objects/entities usually have identifiable intrinsic properties which attribute them as members of a particular type or sort. For instance, in a banking environment the kinds of objects which we may wish to talk about are account numbers, safe deposit boxes, etc. In this context the abstract notion of a safe deposit box refers to the class (or sort) of all objects that can be said to be safe deposit boxes, whereas particular boxes are instances of this sort. As we shall soon see, introducing this typing structure into our framework does not disturb things too much, it is simply a convenience. Indeed it is always possible to translate back to an unsorted universe, the details of which may be found in [En72].

The apparatus of many sorted first order logic has been well exploited in the specification arena both as a tool for describing systems and as a calculus for reasoning about systems - see for example [MW86] [Le86] and [NHN80]. In common with these approaches we shall use this apparatus as a means of representing what have been termed the static aspects of system specification, that is the properties and relationships between the entities of the system. Of course most systems are not simply static and we must also consider how systems change. Our approach will be to introduce the notion of an action which we shall consider to be the mechanism through which systems exhibit change.

Our intuition about actions is similar to most imperative or state based paradigms, where actions are characterised by stating properties which hold before and after an action is performed. The notion of a state is usually introduced to lend a formal semantics to this characterisation. This idea of a "state based semantics" is now commonplace, its use both as a description of a fragment of reality and as a basis upon which to reason about change has been well exploited: specification languages such as VDM and Z, programming languages such as Smalltalk [Smal], and A.I. formalisms such as Planner [He69], STRIPS [FN71] and derivatives. As a consequence its intuitive meaning has been much overloaded and confused.

To us, and we suspect others, the term "state" immediately conjures up phrases such as "state of computation" and imagery of other stable physical situations. In fact the word "state" is often associated with the word "static". Consider, for example, database theory

where *static* integrity constraints are used as a way of characterising valid database *states*. Now, think of any real time concurrent system and try to think of it as moving from one state to another. If the state of such a system refers to some stable/static situation, then it is quite possible that the system will never be in such a condition. If, on the other hand, we are to think of system states as idealised situations in which we can somehow 'freeze' the system and point to all the relevant properties of that frozen situation, why not come clean and make a distinction between system states and the corresponding idealised versions used in most specification languages/formalisms. We think this can best be done by regarding the latter as states of information about a system and have coined the term "scenario" to denote this.

Although this may seem like a distinction of little practical consequence it can drastically effect the way in which we look at the description of dynamic systems. Take, for example, the problem of stating the collision handling properties of a network, in particular, the problem of stating what happens when two nodes try and use the network "at the same time". What is the notion of system state in this situation? Surely there is not one, simply a continuum during which the collision takes place; in such situations all we can possibly refer to is the information about the state of the network, namely that it is in a collision situation. The same goes for most concurrent systems where at any observation instant there will be no notion of global system state. However, even if such is the case, we still have information about the system, namely that certain actions are occurring. It would seem then that the move towards information states and away from system states takes us one step further from associating the description of how a system can change with how in fact that change takes place.

The idea of scenarios as opposed to states is also significant if we are to deal with applications about which we do not have complete knowledge. It is a commonly accepted **tenant** of specification philosophy that a specification may only be judged 'complete' if it captures all the properties we intend the system in question to exhibit. This may involve remaining silent on some properties, asserting others and denying still others. We think that this should also hold true of system scenarios and our model of a system specification should reflect this directly. To this end we have decided to regard scenarios as theories rather than truth functional states. Hence, in general, a scenario will correspond to a collection of truth functional states - each being a completion of the information that that scenario does not decide upon. A further consequence of this approach is that it enables us to represent negative information about actions, for

example "it is not the case that this action produces property ϵ ", without concluding that such an action will result in the negation of that property, that is $\neg \epsilon$.

Although the notion of a scenario is central to our formalism, it is an implicit device which is never totally described; instead, we are only ever indicative of scenarios via the information we have about action descriptions. In contrast to this the idea of a state in formalisms such as VDM and Z is an explicit structure which is carried about effecting and in turn being effected by the performance of actions. When we discuss the role of frame rules we shall see why, like them, we can only take this view when we talk about a specific model of a specification and why it is not possible (from a practical point of view) to consider such things at a specification level.

The introduction to this chapter would not be complete without saying something about the opponents of the state based or imperative approach to specification. We are of course referring to the algebraic school - a pragmatic manifestation of which have become known as the functionalists. The first thing to note is that in recent times many algebraists have found it useful to incorporate the concept of state into their work, witness the work of [LEG85] and [We79] in the field of database specification and [SFS88] in the area of specification. Having said this, state based approaches are still criticised, particularly by those researchers who advocate the abstract data type approach to specification.

Their objection to state based approaches is based, we believe, and with this we sympathise, on the view that a system specification should be independent of that system's implementation states. Like us the algebraists see a specification as providing an information structure upon which particular implementations may be constructed. Compare this with other state based approaches which require the description of an explicit state before sense can be made of imperative operations. This forces them to drop to a model of a specification almost immediately. Note that we do not have any quarrel with this *per se*, indeed it is very important if we are to build tools to support the validation, verification and completeness of specifications, but feel that states should - in the first instance - be an implicit device to be introduced solely to support the characterisation of actions, and not define the structure of intended system states.

One advantage that our approach has over the algebraic school is that we can distinguish clearly between those operations which have an effect on this information structure and

those which do not. To illustrate this point consider the familiar *Top* and *Push* operations of a stack. *Top* is an operation which when performed does not change the information we have about the stack, *Push* on the other hand does. In the algebraic setting both these operations are uniformly modelled by functions. In our formalism *Push* would be modelled as an action, something that describes *change* whereas *Top* would be represented by a function whose extension would describe information *about* a (particular scenario of a) stack. In all fairness it should be pointed out that one can distinguish between the *Top* and *Push* operations in an algebraic specification, in an implicit sense, by capitalising on the type (or sorts) of the functions, e.g. the operation *Top* would return an integer and the *Push* operation a stack. Our complaint is that this is too implicit an approach and is by nature an application dependent process - we can only identify "actions" by an analysis of the application dependent sorts. For a more detailed discussion of this argument the reader is referred to [KMS86] and more recently [KMS87].

Another potential source of objection to our approach may stem from our adherence to a global state setting - especially by those researchers interested in modelling concurrent systems. In response to this dislike of global states some researchers have advocated the introduction of local system states to deal with the problem of stating concurrent system requirements. We believe, that in the first instance, especially whilst dealing with the *requirements* of a system, it is often advisable to remain within a global "state" setting. There are two main reasons for this belief. Firstly, it seems to us that the allocation of local states, that is, the identification that there will be one or more independent sub-systems, and in particular which ones are chosen, is a design decision which we may not necessarily want to contend with at the level of trying to capture a system's requirement specification. Secondly, it is often difficult to assert some requirements which by their very nature demand reference to the entire system. An example of these kinds of requirements is found in the area of protocol specification where certain liveness and safety properties need to be asserted. See [D-H86] for a discussion of the benefits of a global state approach to protocol specification; here the author suggests a top down refinement methodology starting from an initial global view of a network service.

Again we are not arguing against the notion of local states *per se* , but we do feel, that, in the first instance, requirement specification formalisms should adopt a system wide approach leaving it to subsequent steps, such as high level abstract design, as to whether

or not independently operating sub-systems are warranted. Later (in the fifth chapter of this dissertation) we introduce the idea of an agent and show how this can be used to identify potentially independent subsystems.

The remainder of this chapter is structured as follows: first, a formal presentation of the logic SFOAL (short for Sorted First Order Action Logic) will be given, including the definition of a SFOAL theory. This is followed by an intuitive account of the meaning of various SFOAL formulae, together with what constitutes a specificand in this logic. Next we give the definition of a model for a specificand written in SFOAL and end the chapter by discussing a SFOAL specification of the requirements of a South American shipping company.

2.2 SFOAL (*Sorted First Order Action Logic*).

There are two aspects to defining any logic: the idea of a language and the description of a deductive apparatus which characterises the kind of "reasoning" afforded by the logic. The former is usually provided by defining some basic syntactic categories, on top of which some formation rules are stipulated in order to distinguish a special set of syntactic expressions called formulae. The latter is usually given by a collection of axioms and/or a set of rules (of inference; intuitively these can be thought of as licensing the derivation of certain formulae from others). We have decided, for reasons which are made clear later, to adopt a proof theoretic approach to the description of the deductive apparatus of SFOAL. The language is presented first, followed by our proof theory, which describes the formulation of a proof and introduces a provability relation $\vdash$ which signifies when certain formulae are provable from others. We end by defining a SFOAL theory.

LANGUAGE

Syntactic Categories:

- A non empty finite collection of sorts $S = \text{Act} \cup \{ s_1, s_2, \dots \}$, where Act denotes a non empty action sort.
- Constant symbols: for each sort $s \in S$ there is a set (possibly empty) of constant symbols each of which is said to be of sort s .
- Action names: for each $n > 0$ and each n -tuple, $\langle s_1 \dots s_n \rangle$ such that $s_1 \in S, \dots, s_n \in S$, there is a set (possibly empty) of n -place action names, belonging to ACT (of sort ACT) each of which is said to be of type $\langle s_1 \dots s_n \rangle$.
- Predicate symbols: for each $n > 0$ and each n -tuple, $\langle s_1 \dots s_n \rangle$ such that $s_1 \in S, \dots, s_n \in S$, there is a set (possibly empty) of n -place predicate symbols each of which is said to be of type $\langle s_1 \dots s_n \rangle$.
- Function symbols: for each $n > 0$ and each $n+1$ -tuple, $\langle s_1 \dots s_n, s_{n+1} \rangle$ such that

$s_1 \in S, \dots s_{n+1} \in S$, there is a set (possibly empty) of n -place function symbols each of which is said to be of type $\langle s_1 \dots s_{n+1} \rangle$.

- Equality symbols: for certain (possibly all) sorts $s \in S$ there may be a special predicate symbol $=_s$ of sort $\langle s, s \rangle$ representing equality between objects of sort s .
- Variables: the usual infinite collection of distinct variables for each sort $s \in S$.
- Quantifiers: for each sort $s \in S$ we have the universal quantifier $\forall_s$ and the existential quantifier $\exists_s$.
- Logical operators: $\neg, \leftrightarrow, \rightarrow, \wedge, \vee$ and $[]$.
- Punctuation: $(,),$ and $.,$

Formation Rules:

- Terms:
 - for each sort $s \in S$ a variable or constant of sort s is a term.
 - if $t_1 \dots t_n$ are terms of sorts $s_1 \dots s_n$ respectively and $s_1 \dots s_n$ are all taken from S and f is a function symbol of type $\langle s_1 \dots s_n, s_{n+1} \rangle$ then $f(t_1 \dots t_n)$ is a term of sort s_{n+1} .
 - if $t_1 \dots t_n$ are terms of sorts $s_1 \dots s_n$ respectively and $s_1 \dots s_n$ are all taken from S and a is an action symbol of type $\langle s_1 \dots s_n \rangle$ then $a(t_1 \dots t_n)$ is a term of sort Act .
 - nothing else is a term.
- Atoms:
 - if $t_1 \dots t_n$ are terms of sorts $s_1 \dots s_n$ respectively and $s_1 \dots s_n$ are all taken from

S and p is a predicate symbol of type $\langle s_1 \dots s_n \rangle$ then $p(t_1 \dots t_n)$ is an atom.

- for each sort $s \in S$, given two terms of s , t_1 and t_2 , $t_1 =_s t_2$ is an atom, provided that $=_s$ is in the language.
- nothing else is an atom.

- **Formulae:**

- an atom is a formula.
- if ϕ is a formula so is $\neg \phi$.
- if ϕ and ψ are formulae then so are $(\phi \vee \psi)$, $(\phi \wedge \psi)$, $(\phi \rightarrow \psi)$ and $(\phi \leftrightarrow \psi)$.
- if α is a term of sort **Act** and ϕ is a formula then so is $[\alpha](\phi)$.
- for each sort $s \in S$, if x is a variable of sort s and ϕ is a formula then $\forall_s x \phi$ and $\exists_s x \phi$ are formulae.
- nothing else is a formula.

This concludes our presentation of the language, except to say that we will retain the convention we have already begun of using the symbols $\alpha, \beta, \gamma, \dots$ etc. as meta variables over action names and $\phi, \psi, \tau \dots$ as meta variables ranging over formulae.

Proofs.

We shall define proofs in terms of a provability relation, using the meta symbol " $\vdash$ ". If $\text{Form}(L)$ denotes the set of formulae in the language of our logic and $P(\text{Form}(L))$ denotes the powerset of this set then the meta relation $\vdash$ is of type $P(\text{Form}(L)) \times \text{Form}(L)$. In order to do this however we shall need to introduce some axioms and rules.

Axioms.

If ϕ , ψ and τ are formulae and α is a term of sort **Act** then the following are axioms.

- (1) $\phi \rightarrow (\psi \rightarrow \phi)$
- (2) $(\tau \rightarrow (\phi \rightarrow \psi)) \rightarrow ((\tau \rightarrow \phi) \rightarrow (\tau \rightarrow \psi))$
- (3) $(\neg \phi \rightarrow \neg \psi) \rightarrow ((\neg \phi \rightarrow \psi) \rightarrow \phi)$
- (4) $\forall x \phi(x) \rightarrow \phi(t)$ where t is free for x in ϕ .
- (5) $(\forall x (\phi \rightarrow \psi)) \rightarrow (\phi \rightarrow \forall x \psi)$ where x is not free in ϕ .
- (6) $[\alpha] (\text{TRUE})$
- (7) $([\alpha] (\phi \rightarrow \psi)) \rightarrow (([\alpha] \phi) \rightarrow ([\alpha] \psi))$.
- (8) $([\alpha] \neg \phi) \rightarrow (\neg [\alpha] \phi)$.
- (9) $\forall x ([\alpha] \phi) \leftrightarrow ([\alpha] \forall x \phi)$ where x is not free in α .
- (10) $(([\alpha] \phi) \wedge ([\alpha] \psi)) \leftrightarrow ([\alpha] \phi \wedge \psi)$.
- (11) $(([\alpha] \phi) \vee ([\alpha] \psi)) \rightarrow ([\alpha] \phi \vee \psi)$.
- (12) $\exists x ([\alpha] \phi) \rightarrow ([\alpha] \exists x \phi)$ where x is not free in α .
- (EQ) $\forall x, y ((\phi(x) \wedge x=y) \rightarrow \phi(y))$ where y is not within the scope of a modal operator.

Note that $\exists x$ is shorthand for $\neg \forall x \neg$, and **TRUE** denotes truth, and is representative of any tautology of SFOAL e.g. $\phi \vee \neg \phi$. The constant **FALSE** is then defined as $\text{TRUE} \leftrightarrow \neg \text{FALSE}$. Further, the logical connectives $\vee, \wedge$ and $\leftrightarrow$ are defined in terms of $\neg$ and $\rightarrow$ in the usual way. The last axiom has a side condition on the way equality operates in the scope of actions; it is needed in order to have non rigid designators (constants which may change denotation between scenarios). All the other equality axioms such as reflexivity, symmetry and transitivity are as usual and are omitted here. For the present one may think of (EQ) as constraining the usual laws of equality to operate within each scenario remaining silent about any trans-scenario equality properties. We shall return to this point in the next section.

Rules:

Modus Ponens:

Given a proof of ϕ and a proof of $\phi \rightarrow \psi$,
infer a proof of ψ .

Generalisation:

Given a proof of ϕ ,
infer a proof of $\forall x \phi$.

We are now in a position to characterise $\vdash$. Assuming $\Gamma, \Gamma' \dots$ are meta variables over sets of formulae and ϕ is a formula, then we say ϕ is provable from Γ , written $\Gamma \vdash \phi$, iff there is a finite sequence of formulae each of which is an instance of an axiom or a formula within Γ (usually called an assumption) or is inferable from previous lines by one of the two proof rules modus ponens and generalisation (with the stipulation that the variable in generalisation inferences must not occur free in any earlier assumptions). A formula ϕ is a theorem, written $\vdash \phi$, iff it is provable from the empty set of assumptions. Trivially then, all axioms are theorems. In practice we shall not adhere to this definition of proof, adopting instead a shorter natural deduction style approach. We shall assume that the reader is familiar with this kind of (classical logic) proof technique which can be found in standard logic texts such as [N-S85]. In short the style of proof adopted is nothing but a convenience as the kind of first order natural deduction style manipulation which is used has been shown to be equivalent to the axiomatic formulation shown above. The reader is referred to [Lem65] for such a proof, where he may also find the corresponding natural deduction style rules. Note that all non first order manipulation is achieved via the application of the appropriate axioms. The interested reader is also referred to [CC87] wherein the author has characterised SFOAL in a tableau style proof system.

The informal wording of the above rules, Generalisation and Modus ponens may now be stated as

$$\begin{array}{ccc} \frac{\vdash \phi}{\vdash \forall x \phi} & \text{Gen} & \frac{\vdash \phi \text{ and } \vdash \phi \rightarrow \psi}{\vdash \psi} \quad \text{M.P.} \end{array}$$

where the ruled line is to be read " if *top* then *bottom* ".

The anti-extension of the meta proof relation " $\vdash$ " is symbolised by " $\perp$ " and is to be read "not provable"; it is defined as follows $\perp \varphi \leftrightarrow \text{not } \vdash \varphi$. This notion of not provable must be distinguished from the notion of proving the negation of some expression i.e. $\perp \varphi \leftrightarrow \vdash \neg \varphi$. This difference will allow us to refer explicitly to incomplete information, liberating us from the need to make the assumption that given any formula which we cannot prove, we can prove its negation. We do however retain the (consistency) property which says that if we can prove a formula's negation we cannot prove that formula i.e. $\vdash \neg \varphi \rightarrow \perp \varphi$. When we come to talk about reasoning about actions we shall see that this consistency property is preserved by actions, c.f. axiom (8) above.

Theories

We shall introduce the notion of a theory in the above logic via the notion of a theory presentation.

A theory presentation TP , is given by a pair $\langle L_{TP}, A_{TP} \rangle$ where the first component denotes the extralogical language of the theory, such as theory specific sort, predicate and function names, used to formulate a collection of theory specific axioms, denoted by the second component. We can think of L_{TP} as providing concrete syntax for the schematic syntactic categories of our theory.

A SFOAL theory TH of a theory presentation TP , denoted $TH(TP)$, is defined to be the set of formulae which are provable from the theory specific axioms A_{TP} . This can be expressed as

$$TH(TP) = \{ \varphi : A \vdash \varphi \}$$

where φ is a formula expressed in the language L_{TP} .

Note well, we shall often label some proof steps FOL, it will be assumed the reader is familiar with these first order logic manipulations, such as conjunction elimination and De Morgan's laws..This concludes our presentation of SFOAL; we turn now to the utility of such a logic as a formalism for expressing system requirements.

2.3 SFOAL Specifications.

In the introduction to this chapter we noted that there were essentially three aspects to formalising the structural properties of a system: an analysis of the kinds of entities that go to make up the system, a way in which to describe the properties of and relationships between entities, and finally a means by which to capture the description of actions. The remainder of this section deals with how SFOAL can be used to capture these three structural facets of system requirements.

We will divide our discussion of SFOAL formulae into essentially three categories, first order, atomic modal and non atomic modal. To add substance to our discussion we shall formulate some of the requirements of a fictitious bank. First let us set the scene by presenting the language (and its intuitive interpretation) for our chosen example domain.

The language below is intended only as a mechanism for the following discussion; it is certainly too impoverished to be representative of any true banking environment. We also make the assumption that primitive well known types such as Integer (as well as associated relationships such as $<$, $\leq$ etc) are available and for reasons of clarity and brevity will not include their description here.

Sorts: Current-accs, Customers and Act.

Constants for:

Current-accs: 101,102,103,104,105.

Customers: Sam, Tom and Jill.

Act: none.

Predicates:

Incredit: of type $<$ Current-accs $>$.

Functions:

Amount: of type $<$ Current-accs, Integer $>$.

Account: of type $<$ Customer, Current-accs $>$.

Action names:

Debit: of type $<$ Current-accs, Integer $>$.

Credit: of type $<$ Current-accs, Integer $>$.

There are two basic types of entities which feature in our banking example, Current-accs and Customers. To keep matters simple I have restricted the value of bank accounts to have integer value, this is not a major concession as the conventional method of recording sums of monies is in currency units and not 'real' numbers. The predicate Incredit signifies those accounts whose value is above nil (zero value) and the functions Amount and Account denote the amount of monies held in a account and the account number of a customer, respectively. The actions Debit and Credit constitute the obvious means by which accounts can be debited or credited. Although we shall try as much as possible, in the ensuing discussion, to stick to the above language we may use addition constants etc. wherever necessary.

Before we continue it should be noted that the formulae that follow are to be seen as potential axioms of a SFOAL theory, using the language above. We shall delay defining the relationship between a SFOAL theory and a specification until the next section.

First order formulae.

This collection of formulae consists of all those expressions formed using the constructs and formation rules of SFOAL with the exception of the modal (or rather action) connectives "[_] _". In general, first order formulae are used to express the static features of our domain. Consider for example the following:

$$\forall c, c' (\text{Account}(c) = \text{Account}(c')) \rightarrow c = c'.$$

The above formula expresses the property that any two distinct customers cannot have the same bank account. If we were in the business of setting up a database to record customer accounts, then this formula would be considered to be something called a static integrity constraint or alternatively a state invariant. We shall return to this point and a related one, about integrity, when we look at the description of actions.

The reader will note that we have not strictly adhered to the formation rules of SFOAL, in so much as dropping various subscripts and leaving out some punctuation symbols such as brackets, where these are clear from context. This is a practice of convenience which we shall adopt from now on. In reading such expressions the reader is reminded of the standard binding precedences of the logical connectives in first order logic - which apply equally to the many sorted version used here. The binding strength of the modal connectives will be stated when we discuss modal formulae.

Less general properties can also be stated, for example,

$$\text{Incredit}(\text{Account}(\text{Sam}))$$

$$\text{Account}(\text{Sam}) = 101$$

These state that the customer Sam is in credit and his account number is 101, respectively. We may also require that each customer must have an account, expressed for example by

$$\forall c \exists a \text{ Account}(c) = a.$$

This particular requirement brings out an interesting point about quantifiers. The point rests on the difference between potential and actual domains; for us, the quantification over the customer and account domains refers to potential domains. Let us explain what we mean by this. When a sort is declared it is not intended to represent a particular collection of actual objects, customers say, instead it is meant to denote the class of all potential objects which may become customers. The role of sorting or typing is therefore an intensional device and so quantification over sorts is also intensional. In order to quantify over specific collections of objects we can then qualify these intensional quantifiers. The customer sort provides us with a good example of this; originally we had three constants of sort customer but this was more a convenience than anything else. In a real banking environment the notion of a constant customer is rare indeed. What is more usual is that a person is free to open and close accounts at a particular bank, i.e. the number of actual customers is a contingent value. This more realistic situation may be dealt with by introducing a predicate "Client" whose extension varies as does the domain of actual customers, the change in the extension being governed by the occurrences of the actions "open" and "close" account. Hence if we wished to state some property ϕ about "current" customers - or should we say clients - we would write

$$\forall c \text{ Client}(c) \rightarrow \phi.$$

To bring this point closer to home we only need to consider specifications which include the integer sort; here it is commonly accepted that any implementation of such specifications will only include some subset of the integers; similarly a particular bank will only keep records of a finite collection of customers. The point though is that this finite collection is contingent in time and may either decrease or increase; rather than adopting domains which vary (an extremely difficult solution if domains are allowed to decrease) we have adopted this idea of a potential domain, discriminating objects of interest by the information we have of them. We shall return to this point about

quantification when we discuss the meaning of quantifiers in the context of modal formulae.

Since we shall employ this method of restricting the scope of quantifiers often it will be worth providing its general form - which is

$$\forall x \text{ Rel}(x) \rightarrow \phi$$

where the universal quantifier refers to the same sort as the variable x ranges over, $\text{Rel}(x)$ denotes the collection of objects of sort " x " having property " Rel " and ϕ is a formula which expresses what we wish to say about the collection of objects of sort " x " having property " Rel ". The expression $\text{Rel}(x)$ is known as a relativisation predicate; note that the expression $\text{Rel}(x)$ need not be a simple (i.e. atomic) predicate.

As a final example of a first order formula consider the following expression

$$\forall a \text{ Amount}(a) \geq 0 \rightarrow \text{Incredit}(a)$$

which states that any account holding a non negative amount is considered to be in credit. Note that this still allows for the possibility of an account being in credit even when that account holds a non positive amount. This may for instance occur when a bank manager permits an overdraft.

Before going on to discuss actions I would like to say something about the way in which the syntactic category **Act** features in first order expressions. Perhaps the first observation to be made is that objects of sort **Act** are action names not the actions themselves. The way in which we talk about, or characterise, actions is via the modal connectives " $[_]$ " and is the subject of the next section. As can be seen from the formation rules of SFOAL, action names are allowed to act as arguments to predicates and functions, as well as being the results of functions. So relationships between actions are expressed by stating relationships between their names (and we shall often refer to action names as actions).

Having the facility to predicate on action names is very useful. If for instance we made available a predicate of type $\langle \text{Act}, \text{Act} \rangle$ called Not-together we may in the context of our banking example use it to state the property that no bank account can be both credited and debited at the same time. In symbols we would write

$$\forall a \text{ Not-together}(\text{Debit}(a), \text{Credit}(a))$$

where 'a' is a variable ranging over customer's bank accounts. As well as allowing relationships to exist between actions, the facility of predicating action names will also allow the structuring of the action universe **Act**. For instance we could introduce a sub-sorting hierarchy between actions, using the notion of a relativisation predicate, as discussed above. The authors of [We83] and [Ba81] suggest this kind of structuring for the abstract specification of databases.

Allowing functions the ability to return and accept action names fulfills a variety of purposes. For instance we can express the requirement that an assistant bank manager, named Fred, is responsible for all of Tom's debit transactions by the following expression

$$\forall x \text{ Responsible(Debit(Account(Tom), x)) = Fred.}$$

The ability to construct expressions such as the above also provides a basis for more general management accountability structures. In any commercial organisation there will be some sort of management structure, part of which will deal with the problem of apportioning responsibility as regards certain actions. As further evidence of the usefulness of having actions as the arguments and results of functions consider the following functions, each of which returns the appropriate "response" action to a certain prevailing state of affairs or context:

$$\text{Response(system-error) = restart}$$

$$\text{Response(bus-interrupt) = service-bus}$$

$$\forall c \text{ Response(} \neg \text{ Incredit(c)) = mail-reminder(c) ...etc.}$$

The idea is to provide a basic set of actions **Act** which may stand in certain relationships to each other, using predicate and function names. The ability to parameterise functions by action names can also facilitate the generation of non primitive actions. Consider for example a function which takes as argument two action names and returns an action, the returned action being the sequential composition of the two argument actions. Then this function could be used to construct more complex actions from primitive ones. Although I shall introduce some *logical* action combinators, such as sequential composition, there is no reason why extralogical action connectives (those which are specific to a particular application) cannot be properly expressed using functions. We shall return to this issue when we introduce action combinators in chapter four.

One last point remains to be made in connection with the syntactic category **Act**,

namely, that we have purposely admitted the ability to parameterise action names by action names. The decision to do so was based upon the observation that it is often useful to formulate actions which need to *mention* the names of other actions. To illustrate this point, suppose that we were describing an action called "send-message" and the purpose of this action was to display at a remote site the fact that a particular action was being performed. Then the action "send-message" might have as parameter an action name, namely the action whose performance is being reported. This ability to treat elements of the sort **Act** as terms reinforces the difference between the *use* of terms and their *mention*. The appearance of action terms within the modal (and as we shall see later deontic) contexts are particular uses to which action names are put.

Atomic modal formulae.

A modal formula is one in which the modal connectives "[_] _" appear. An atomic modal formula is one which has the syntactic form

$$[\alpha] \phi$$

where α denotes a term of the action universe **Act** and ϕ denotes a first order formula as discussed above. Classically, in fact as far back as Aristotle [HC68], the term modal is used to indicate change, that is the movement between modes; for us a mode shall be a scenario and an atomic modal formula will express some property about the movement between scenarios. Hence, before we proceed further it would be advantageous to explain in more detail our notion of scenario.

Scenarios.

For the reasons given in the introduction to this chapter, we have decided to base our description of actions on the notion of a global state, where a global state is referred to as a scenario. A scenario can be thought of as representing the collection of observable system properties at a given observation instant. We do not mean observation in any physical way but rather in an envisaged sense. For instance, the statement "If the recovery routine is executing, then performing a reboot results in files being lost" is a statement describing the result of an action in an envisaged scenario, i.e. one in which a recovery routine is executing.

In the remainder of this chapter, scenarios will be represented by consistent SFOAL

theories and we use the expressions $t(s)$, $t(s')$, $t(s'')$ etc as meta notation to range over scenarios (where s , s' ... etc to refer to the corresponding theory presentations). In addition to this where we had previously defined provability with respect to a SFOAL theory presentation $\langle L, A \rangle$ as the relationship $A \vdash \phi$, read ϕ is provable from $\langle L, A \rangle$, we shall from now onwards employ the shorter symbolism $\vdash_s$ where s is the theory presentation ($\langle L, A \rangle$) of the theory representing the scenario $t(s)$.

There is of course a special relationship between the theories which correspond to scenarios of a system and the theory which is the denotation of the system specification. We shall not say exactly what this is until we come to the topic of specificands and their models. The reader should regard the SFOAL theories corresponding to scenarios as representing the information we have about the state of a system at any given time. Hence, the information content of a particular scenario should be considered to be a contingent matter.

In the context of our banking example there are many possible scenarios which may be envisaged. For example in one scenario Tom could be creditworthy, and we could reflect this by attributing to his bank account the property of being "Incredit", whereas in another he may well be in debt. We can represent this information, in SFOAL, by being indicative of the two distinct scenarios depicting Tom's credit status, i.e.

$$\vdash_s \text{Incredit}(\text{Account}(\text{Tom}))$$

$$\vdash_{s'} \neg \text{Incredit}(\text{Account}(\text{Tom}))$$

Note that it is not possible for both of these properties to belong to the same scenario, since inconsistency would arise. Having introduced scenarios we are now in a better position to talk about actions, or rather their description, expressed as atomic modal formulae.

Describing actions.

The description of an action in SFOAL is based upon characterising the hypothetical movement between scenarios. An atomic modal formula $[\alpha] \phi$, where α is an action name and ϕ is a first order formula, if provable from a particular scenario s , states that if the action α is performed in scenario s , then the scenario reached by performing α in s , $\alpha(s)$ say, exhibits the first order property ϕ .

We can state this more succinctly by the (informal meta) rules $\mathcal{M}$ and $\mathcal{M}'$,

$$\frac{\vdash_s [\alpha] \varphi}{\vdash_{\alpha(s)} \varphi} \quad \mathcal{M} \qquad \frac{\vdash_s \neg [\alpha] \varphi}{\vdash_{\alpha(s)} \varphi} \quad \mathcal{M}'$$

We shall from now on retain the convention of naming by $\beta(s)$ the scenario reached by performing an action β in a scenario s . This usage is purely syntactical and provides us with the means to talk about those scenarios reached by performing specific actions in other scenarios. We repeat that $\mathcal{M}$ and $\mathcal{M}'$ are not rules of our logic, merely convenient devices for intuitive explanation.

If we look a little closer at this rule we discover that it is a very weak characterisation of movement. Taken literally, all it demands is that whenever an action α is performed in a scenario s then the resulting scenario $\alpha(s)$ exhibits the explicitly stated consequence of that action. This however is not sufficient to capture our full intent.

As an illustration of its failing, consider a scenario s where the amount of Sam's bank account is in the red by 200 units, but has been agreed with the bank as an overdraft and he is still therefore considered to be in credit. Further, in the absence of any general rule, we shall also assume that if he deposits 100 units the amount outstanding will be 100 units. This may be represented symbolically as:

$$\begin{aligned} &\vdash_s \text{Incredit}(\text{Account}(\text{Sam})) \text{ and} \\ &\vdash_s \text{Amount}(\text{Account}(\text{Sam})) = -200 \text{ and} \\ &\vdash_s [\text{Credit}(\text{Account}(\text{Sam}), 100)] \text{Amount}(\text{Account}(\text{Sam})) = -100. \end{aligned}$$

Then, according to the movement Rule, $\mathcal{M}$, we have

$$\vdash_{C(s)} \text{Amount}(\text{Account}(\text{Sam})) = -100$$

where $C(s)$ is shorthand for the action of crediting Sam's account by 100 units. However, we would also like to preserve the property that Sam is still in credit until something happens which explicitly alters this fact, such as the bank manager withdrawing overdraft facilities. But our $\mathcal{M}$ rule is noncommittal either way; accordingly the state $C(s)$ may or may not preserve certain properties of the state s .

In general, when an action occurs in the real world the corresponding change preserves all those properties of the world which are not affected by that action. So, in a situation

in which the door and window of a building are closed, opening the window should not in general also cause the door to open. Rather, the door should remain closed; in other words, actions have an implicit scope. To reflect this observation explicitly in our logic would involve stating not only the effects of an action but also those things (properties) which are not affected by it.

There are a number of reasons why such an explicit approach is not desirable, perhaps the most important of which has pragmatic foundation. It would be unreasonable to expect for example that in the process of describing the action of closing a door we were also obliged to state all the passive effects of performing this action: the window does not open, Tom's salary does not increase, etc. As we can see this list is potentially never-ending and even if in a very closed domain all the properties of performing an action could be stated, would this not blur the all important clarity criteria that most specification approaches - ours included - seek ?

In addition to this, when we describe actions we may wish to be purposefully vague, either explicitly or implicitly, leaving to a particular implementation what the exact nature of the action is. This point is especially true of requirement specifications where the exact nature of an action may not as yet be known. As an illustration of this consider the following state of affairs:

- (1) $\vdash_s \text{Incredit}(\text{Sam})$
- (2) $\vdash_s \text{Incredit}(\text{Sam}) \rightarrow \text{Rich}(\text{Sam})$
- (3) $\vdash_s [\text{Spendalot}(\text{Sam})] \neg \text{Rich}(\text{Sam}).$

When the action "Spendalot(Sam)" is performed in the scenario s , we can use $\mathcal{M}$ to conclude that the resultant scenario, $S(s)$ say, has the property that Sam is no longer rich. But we cannot consistently preserve both of the other properties, in other words both $\vdash_{S(s)} \text{Incredit}(\text{Sam})$ and $\vdash_{S(s)} \text{Incredit}(\text{Sam}) \rightarrow \text{Rich}(\text{Sam})$ cannot hold. We are in fact faced with a choice - we can preserve either (1) or (2) or neither - this choice depends on how we choose to implement the action "Spendalot(Sam)". We shall refer to actions of this kind as being implicitly underdetermined.

This same phenomenon can be seen when we wish to be explicitly vague about the outcome of an action. Consider an action description of the form $\vdash_s [\alpha] \phi \vee \psi$. By our rule $\mathcal{M}$ we can conclude that $\vdash_{\alpha(s)} \phi \vee \psi$. But again a choice is involved in determining the precise nature of the action α , since there are at least four different scenarios that are

acceptable, as far as our $\mathcal{M}$ rule is concerned:

$$\begin{aligned} & \vdash_{\alpha(s)} \varphi \text{ and } \vdash_{\alpha(s)} \psi \\ & \vdash_{\alpha(s)} \psi \text{ and } \vdash_{\alpha(s)} \varphi \\ & \vdash_{\alpha(s)} \psi \text{ and } \vdash_{\alpha(s)} \varphi \text{ and finally} \\ & \vdash_{\alpha(s)} \psi \vee \varphi, \vdash_{\alpha(s)} \psi \text{ and } \vdash_{\alpha(s)} \varphi. \end{aligned}$$

Although the latter kind of underdeterminism, the explicit kind, is catered for in most specification formalisms, the implicit kind is usually dealt with at an informal level. Most specification formalisms do this by implicitly assuming that things remain as they are, unless an action explicitly changes them. This implicit requirement, that any property that an action does not effect should be preserved is known as the frame axiom/rule. Indeed most specification languages incorporate some syntactic device for restraining "in an informal way" the effects of actions. Both VDM and Z use blanket term phrases to denote this kind of frame condition. More recently the two tiered specification language Larch [Win83] also restrains the normal/intended effects of actions by asserting their scope and possible side effects explicitly.

In a sense these languages need to do this with more urgency than ourselves because their notion of state is an explicit entity and actions are seen as explicit transformations of states. This is not to say that we do not suffer from the same problems that the incomplete descriptions of actions raise, simply that we regard such "problems" as having solutions only in particular models of a specification.

Non Atomic Modalities.

There are two topics we wish to cover under this heading: firstly, the relationship between first order and modal formulae and, secondly, the concept of iterated modalities.

First order relationships

In our discussion of atomic modal formulae it was often the case that a number of modal properties were observable at a particular scenario s . We represented this as $\vdash_s [\alpha] \varphi$, $\vdash_s [\alpha] \psi$, $\vdash_s [\beta] \varphi$...etc. If however there are a number of atomic modal formulae about the same action which are provable from a common scenario then there may exist

equivalent representations. Consider for example a scenario s such that

$$\vdash_s [\alpha] \phi \text{ and } \vdash_s [\alpha] \psi$$

then we clearly also have

$$\vdash_s [\alpha] \phi \wedge [\alpha] \psi$$

But we can go one step further than this simple observation and note that what the above information about action α in scenario s tells us is that after performing α both ϕ and ψ will hold in the resulting scenario. Another way of stating this would be $\vdash_s [\alpha] (\phi \wedge \psi)$; in fact, all things considered, there should be an equivalence between

$$\vdash_s [\alpha] \phi \wedge [\alpha] \psi \quad \text{and} \quad \vdash_s [\alpha] (\phi \wedge \psi),$$

since they seem to say the same thing.

The laws which govern, in general, the interaction between first order and modal formulae are expressed by axioms seven through twelve of SFOAL. We shall now look at each of these in turn.

Axiom seven of SFOAL,

$$7. \quad ([\alpha] (\phi \rightarrow \psi)) \rightarrow ([\alpha] \phi \rightarrow [\alpha] \psi),$$

states how material implication behaves with respect to modalities. Hence, if an action results in a scenario in which $\phi \rightarrow \psi$ holds, then it follows that if that same action results in ϕ , it must also result in ψ . Note that if this were not the case then it would be possible to perform an action which resulted in a scenario in which $\phi \rightarrow \psi$, ϕ and $\neg \psi$ all hold, which would contradict the rule of modus ponens for that scenario. In short, then, axiom seven preserves the law of modus ponens from scenario to scenario.

However, the reverse of axiom seven is not valid and the reason for this is connected to our view that scenarios should, in general, denote incomplete information states. Accepting the reverse of seven would force complete information in every scenario and allow us to prove the validity of the reverse of the next axiom - something which we now argue against.

Axiom eight deals with negation, asserting

$$8. \quad ([\alpha] \neg \phi) \rightarrow (\neg [\alpha] \phi).$$

This amounts to saying the following: "If it is the case that after performing the action α we reach a scenario in which $\neg \phi$ is provable, then it can never also be the case that performing action α results in a scenario in which ϕ is provable". If this axiom were invalid it would be possible to perform actions which lead to scenarios in which contradictions could be proven; in other words, inconsistent scenarios would become admissible. We can see this more clearly if we note that the only way in which 8 can be invalid is if $[\alpha] \neg \phi$ holds and $\neg [\alpha] \phi$ does not. In addition if $\neg [\alpha] \phi$ does not hold, then - because of excluded middle - $[\alpha] \phi$ holds, and this together with $[\alpha] \neg \phi$ describes an action which leads to a scenario in which both ϕ and $\neg \phi$ hold. Hence, given that some property ϕ does not hold after performing an action, and in order to preserve consistency, we must conclude that the same action does not also result in ϕ .

But, if we are to allow scenarios in which there is only partial information, that is those in which it is possible that neither ϕ nor $\neg \phi$ hold, then we cannot form an equivalence in axiom 8 by adding

$$8'. (\neg [\alpha] \phi) \rightarrow ([\alpha] \neg \phi)$$

Since this would demand that any action not resulting in some property ϕ had to result in $\neg \phi$. This is easier to see if we rewrite 8' using negation and disjunction:

$$8'. ([\alpha] \phi) \vee ([\alpha] \neg \phi)$$

Note that this is different from the valid disjunction $([\alpha] \phi) \vee \neg ([\alpha] \phi)$. This difference will be important when we look at disjunction later.

The manipulation of universal quantifiers is dealt with by axiom nine. Remembering that a universal quantifier ranges over a domain of potential objects, then

$$9. \forall x ([\alpha] \phi) \leftrightarrow ([\alpha] \forall x \phi) \text{ where } \alpha \text{ is not free for } x,$$

says that this potential domain is common to all the scenarios that can be reached by performing actions. Indeed the right to left implication can be taken to be a formulation of the famous Barcan formula which characterises those systems of modal logic in which possible worlds have constant domains; see [HC68] for a discussion of such logics. Hence, in our banking example, the domain of potential customers is common to all scenarios, but the domain of clients may change from scenario to scenario as the opening and closing of accounts dictates. We have already seen the reasons for adopting this view.

An important observation to note about our system is that quantification over actions, or rather action names, is dealt with in the same way as quantification over any other sort. However, because action names are allowed to have parameters we need to be careful about what we mean by quantifying over actions. Considering the above declared Credit action, the expression

$$\forall c ([\text{Credit}(c,100)] \text{ Bonus}(c))$$

says that any customer whose account is credited with 100 units gets a bonus. Here the result of quantifying over the customer sort has the effect of quantifying over many separate actions, in fact over all action name instances of the action type Credit whose credit value is 100 units. We can quantify over the whole action universe by the use of variables in the usual way. Hence the following expression

$$\forall \alpha [\alpha] \phi$$

where α is a variable over the action category Act, states that the property ϕ is invariant with respect to change. This is a very strong global constraint and is an extreme example of what is usually referred to as a dynamic constraint. Weaker dynamic constraints may be expressed by conditionalising the above expression; thus the oft quoted (dynamic) constraint "salaries must not decrease" may be captured thus

$$\forall \alpha \forall x \forall m \text{ Sal}(x) = m \rightarrow [\alpha] \text{ Sal}(x) \geq m$$

where $\text{Sal}(x) = m$ is read as "the salary of person x is m ". Even weaker dynamic constraints can be expressed by using relativisation to restrict dynamic constraints to specific collections of actions. Consider for example the expression

$$\forall \alpha \exists y \text{ Popular}(\alpha) \rightarrow [\alpha] \text{ Penalty}(y)$$

which says that performing any popular action always incurs at least one penalty.

The next axiom, axiom ten,

$$10. ([\alpha] \phi \wedge \psi) \leftrightarrow ([\alpha] \phi \wedge [\alpha] \psi)$$

deals with conjunction. The intuitive argument for the validity of this expression can be divided into two halves, one for each half of the above equivalence. We shall examine the left to right reading first; this states that any action producing the conjunctive effect of two properties can also be described as a conjunction of two action descriptions stating that that action produces both the effects in question.

As an example consider the situation in which, if a consequence of withdrawing a hundred units from Tom's account is that his account does not stay in credit and he is sent a letter from the manager, then one consequence of withdrawing the hundred units should be his insolvency and another that he receives a letter. We can depict this symbolically as

$$\begin{aligned} & [\text{Debit}(\text{Account}(\text{Tom}), 100)] (\neg \text{Incredit}(\text{Account}(\text{Tom})) \wedge \text{Letter}(\text{Tom})) \\ \rightarrow & ([\text{Debit}(\text{Account}(\text{Tom}), 100)] \neg \text{Incredit}(\text{Account}(\text{Tom})) \\ & \wedge [\text{Debit}(\text{Account}(\text{Tom}), 100)] \text{Letter}(\text{Tom})) \end{aligned}$$

where Letter is a predicate of type customer and holds of a customer who receives a letter from the manager.

For similarly intuitive reasons we must also accept that if it is the case that performing any action α , a scenario is reached in which ϕ is provable and it is the case that performing α results in reaching a scenario in which ψ is provable, then it should also be the case that performing α results in reaching a scenario in which both ϕ and ψ (that is $\phi \wedge \psi$) are provable. We would have a very odd system indeed, if this were not the case. In short, axiom ten states that actions distribute over conjunction.

The next axiom deals with disjunction, and also highlights a difference between action descriptions and information about action descriptions. Axiom eleven,

$$11. ([\alpha] \phi \vee [\alpha] \psi) \rightarrow ([\alpha] \phi \vee \psi)$$

as we shall see later, is a derived axiom designed to preserve the disjunctive weakening, $\phi \rightarrow (\phi \vee \psi)$, found in classical logic. The analogous case for actions is $[\alpha] \phi \rightarrow [\alpha](\phi \vee \psi)$ which is obvious if we consider that any action resulting in a property ϕ also - vacuously - results in either ϕ or some other property. Given this it is easy to see the validity of 11.

But the reverse of axiom eleven is not valid in our system; if it were, it would force scenarios to become complete theories. To see this assume the action description $[\alpha] \phi \vee \psi$ which states that performing the action α results in ϕ or ψ ; then, if the reverse of 11 were valid, we would be forced to accept either $[\alpha] \phi$ or $[\alpha] \psi$; but this means that either the action will result in property ϕ or property ψ . Hence, we are left with no room for the case in which the property ' $\phi \vee \psi$ ', as a piece of information in its own right, independent of either ϕ or ψ , can be considered the result of the action α . However, this

situation is accommodated if we reject the reverse of 11 and admit incomplete scenarios (SFOAL theories).

Intuitively, expressions of the type $[\alpha] \phi \vee [\alpha] \psi$ may be seen as information about action descriptions whereas an expression of the form $[\alpha] \phi \vee \psi$ should be seen as *an* undetermined action description. To reinforce the invalidity of

$$11'. [\alpha] \phi \vee \psi \rightarrow ([\alpha] \phi \vee [\alpha] \psi)$$

take ψ to be $\neg \phi$; now since $[\alpha] \phi \vee \neg \phi$ is a tautology of our system (see axiom six of SFOAL) then so to, given the validity of 11', is $[\alpha] \phi \vee [\alpha] \neg \phi$. This may then be rewritten as

$$\neg [\alpha] \phi \rightarrow [\alpha] \neg \phi$$

which is something we argued against earlier.

Axiom twelve of our system is concerned with how the existential quantifier operates in the scope of modalities:

$$12. \exists x ([\alpha] \phi) \rightarrow ([\alpha] \exists x \phi) \text{ where } \alpha \text{ is not free for } x,$$

This can be read as, "If before performing an action we know that at least one object will have property ϕ after performing action α , then after performing that action it should in fact be the case that at least one object has property ϕ ". Indeed for the finite case we can employ the usual disjunctive reading of the existential quantifier to obtain something not unlike the valid disjunctive form of axiom 11,

$$([\alpha] \phi(x_1) \vee [\alpha] \phi(x_2) \vee \dots) \rightarrow [\alpha] (\phi(x_1) \vee \phi(x_2) \vee \dots).$$

Where $x_1, x_2 \dots$ etc denote elements of the domain in question.

But the reverse of twelve is not valid and again we can use the finite case to show this. Assuming that $[\alpha] \exists x \phi$ and a finite domain over which x ranges, we can rewrite this as

$$[\alpha] (\phi(x_1) \vee \phi(x_2) \vee \dots).$$

This means that performing the action α results in at least one object having property ϕ . Now, if the reverse of twelve were also valid, this would also entail $\exists x ([\alpha] \phi)$, which under the same assumptions may be written

$$([\alpha] \phi(x_1) \vee [\alpha] \phi(x_2) \vee \dots).$$

But for reasons we have already seen this would force us to decide which object had property ϕ before performing the action α , whereas the former expression need not force us to decide on *which* object has property ϕ . The reader is referred to the discussion of axiom 11 for further details.

The last axiom is concerned with equality and the admission of non rigid designators. The axiom

$$\text{EQ } \forall x,y ((x=y \wedge \phi(x)) \rightarrow \phi(y)) \text{ where } x \text{ does not appear in the scope of a modal operator.}$$

is a slight variant on the usual substitution of identicals rule. The only difference is that the side condition restricts this to terms which do not appear in the scope of a modal operator. The primary reason for this is the desire to have non rigid designators; basically a non rigid designator is a constant whose denotation is dependent upon the world (or in our case scenario) in which it is evaluated. In other words, non rigid designators can be thought of as constants which change value. As an illustration of this, consider the following expression:

$$\text{Amount} = 50 \rightarrow [\text{Add-10}] \text{Amount} = 60$$

Here Amount is a constant (or nullary function) which is intended to signify the amount of money held in a general account. For the sake of this example we have also chosen the constant action of Add-10, the purpose of which is to credit an account with 10 units. It is quite obvious then that in this situation we intend the constant 'Amount' to change value from scenario to scenario.

However, without the extra side condition in the EQ axiom this would not be possible; consider the traditional substitutional axiom

$$\text{EQ'} } \forall x,y ((x=y \wedge \phi(x)) \rightarrow \phi(y)).$$

This, together with the standard reflexivity laws of equality, would allow us to prove that all actions preserve any equality between terms. In symbols this may be represented as

$$\forall \alpha \forall x,y (x=y \rightarrow [\alpha] x=y).$$

Clearly this is not what is desired. We can see the offending derivation as follows, noting that this not a formal proof, also noting that all variables are universally quantified.

Proof of $\forall \alpha \forall x, y (x=y \rightarrow [\alpha] x=y)$

- | | |
|---|-------------------|
| 1. $x=y$ | Assumption |
| 2. $(x=y \wedge \varphi(x)) \rightarrow \varphi(y)$ | EQ' |
| 3. $[\alpha] x=x$ | Theorem. |
| 4. $x=y \wedge [\alpha] x=x$ | Conjunction 1,3 |
| 5. $(x=y \wedge [\alpha] x=x) \rightarrow [\alpha] x=y$ | EQ' applied to 4. |
| 6. $[\alpha] x=y$ | Modus ponens 4,5. |

This kind of derivation is blocked by not allowing the substitution of identicals property to become a trans scenario property, in other words by restricting this property to be scenario specific; this is achieved by the side condition in axiom EQ. The reader may like to think of this axiom as ensuring the locality (to a scenario) of the notion of equivalent terms.

Iterated modalities.

Although so far we have restricted ourselves to talking about modalities which concern only first order expressions, SFOAL also allows the formation of modalities which concern other modalities. Consider the expression

$$\vdash_s [\alpha] ([\beta] \varphi)$$

where φ may be a modal or first order formula. This is to be read as: "If the action α is performed at scenario s resulting in a scenario $\alpha(s)$, then at $\alpha(s)$ it will be the case that if the action β were to be performed, the resulting scenario, $\beta(\alpha(s))$ say, would exhibit the property φ ". The rule $\mathcal{M}$ spells this out, repeated application bringing about the following sequence:

$$\vdash_s [\alpha] ([\beta] \varphi),$$

$$\vdash_{\alpha(s)} [\beta] \varphi$$

and finally $\vdash_{\beta(\alpha(s))} \varphi$.

As a more concrete illustration of iterated modalities, consider, returning once again to our banking example, the case where after Jill's account is credited if it is credited again without first being debited, then Jill is deemed creditworthy. If, however, Jill's account is debited twice without an intermediate credit transaction being performed, then she

loses her creditworthy status. We could represent this strange, volatile, credit logging activity as

$$\forall x \forall y \ [\text{Credit}(\text{Account}(\text{Jill}), x) \] \ [\text{Credit}(\text{Account}(\text{Jill}), y) \] \ \text{CWorthy}(\text{Jill})$$

$$\forall x \forall y \ [\text{Debit}(\text{Account}(\text{Jill}), x) \] \ [\text{Debit}(\text{Account}(\text{Jill}), y) \] \ \neg \text{CWorthy}(\text{Jill})$$

Note that once Jill's creditworthiness is known, that is, it is provable at a particular scenario, her account can be alternately credited or debited without her credit status being changed. A change will only occur if she is not creditworthy and her account is credited twice, without an intermediary debit, or if she is creditworthy and her account is debited twice without an intermediary credit transaction.

This concludes what we have to say about the intuitive interpretation of SFOAL formulae. Next we shall look at what a specification is in SFOAL and relate the notions of system specification and system scenarios by defining models for SFOAL specifications.

2.4 SFOAL Models.

Before we can begin to discuss models for SFOAL specificands we shall provide a brief intuition as to what such an object is. A specificand is the *object* produced through the *activity* of specification. This activity is often iterative and should be highly structured - see the introduction of this thesis for some remarks about specification methodologies. Here we shall only concern ourselves with the resultant object. In the case of requirement specifications, a specificand can be regarded as a statement of what kind of system is intended (required). If this statement is expressed in SFOAL then the specificand will correspond to a SFOAL theory. Remember that a SFOAL theory is generated by a SFOAL theory presentation which in turn is made up of two parts, an extralogical language and a collection of SFOAL formulae (expressed in this language) called the axioms of the theory. But as we saw earlier a system scenario (information about the particular state of a system) is also represented as a SFOAL theory. There is of course a strong connection between the potential scenarios of an intended system and its specification; the remainder of this section is concerned with describing this relationship. In what follows we shall use the terms specification and specificand interchangeably to denote the object that corresponds to a formal expression of a system's requirements.

Over the lifetime of any dynamic system (one which has the capacity to effectuate change) we will in general be able to observe the system in a number of different states. In our formalism the information associated with these observations is represented by a collection of scenarios. However, when we attempt to capture the dynamic requirements of a system we do not present a collection of scenarios via the formulation of a set of SFOAL theory presentations; this would be impractical to say the least. Instead we try and capture, intensionally, the way in which change occurs. A specificand is this intensional body of knowledge, scenarios are to be seen as particular extensions of this, much in the same way the intensionality of a monadic predicate is realised in different ways depending on what objects have a particular property.

In order to capture this special relationship between a system's specificand and the collection of system scenarios it gives rise to, we need to augment SFOAL with a new rule. Before saying what this rule should be it will help to look at the reason for its existence. We start by remembering that SFOAL is a kind of modal logic where a collection of actions can be seen to be a collection of modalities.

In any system of modal logic some mechanism for the introduction of modal formulae from non modal ones can be found; this usually takes the form of a necessitation rule. The general character of such a rule can be described thus: "If some formula ϕ can be deduced in a system of logic with no extra-logical assumptions, then the formula $\Box\phi$ (read "It is necessary that ϕ is the case") can also be deduced". The idea is that if some property holds (can be proven) independently of any assumptions, then it necessarily holds, which in a modal setting means that it holds in all "possible" worlds. Here the concept of a possible world plays the same role as our notion of scenario, that is it represents a particular state of affairs. Before we attempt to devise a similar rule for SFOAL, two important observations need to be made. Firstly, we must deal with a number of modalities, one for each action, and, secondly, the modalities themselves are dependent on a particular theory of our logic. The first point is not as troublesome as the second.

At first sight it might seem appropriate to augment SFOAL with a rule of the following form

$$\mathcal{N}1 \quad \frac{\vdash \phi}{\vdash [\alpha] \phi}$$

which has more or less the same form as the necessitation rule we mentioned above. What this rule says is that if we can prove ϕ from a particular SFOAL theory then we can also show that performing any action α preserves ϕ . One difficulty with this stems from the fact that scenarios are also represented as SFOAL theories, hence accepting this as a rule of our logic will mean that all properties of all scenarios are preserved by all actions. Clearly this is not a desirable situation, we would like the above rule to be specific only to specifications. This would have the restricted consequence of preserving only the intensional information of the specification and not necessarily the information specific to scenarios.

In order to do this though, we must distinguish between provability from a specification and provability from a scenario. Representing the former as $\vdash_{sp}$, the following more restrictive rule may be stated as

$$\mathcal{N}2 \quad \frac{\vdash_{sp} \phi}{\vdash_{sp} [\alpha] \phi}$$

The first thing to note is that adopting this approach will result in two proof systems, one for scenario based reasoning and one for specificand based reasoning. This causes immense complications if we consider (as we do below) a scenario to be a theory extension of a specification - imagine trying to prove something from a theory part of which is governed by one system of deduction and the remainder by some other. It is obvious that in such a situation the two proof systems will have to be formally related in some way. We can however avoid this complication by taking what we call the static interpretation of a rule; in essence this defines the theory in question to be the smallest set of theorems closed under a particular rule. For the rule above we can define such a theory in the following manner,

$$\text{if } \varphi \in \Gamma \text{ then } [\alpha] \varphi \in \Gamma$$

where Γ is the minimum set of sentences which includes all the theorems of $\text{TH}(\text{sp})$ and satisfies the above rule and the variable α ranges over the action category **Act** of **sp**.

Henceforth whenever we talk about the theory associated with a SFOAL theory presentation for a specification, **sp**, we shall assume that in addition to the theorems derivable in the way we defined earlier (see the definition of a proof from a SFOAL theory) we also have available the theorems in the set Γ as defined above. This has the same effect as augmenting specificands written in SFOAL with the rule **R2**. We are now in a position to define what constitutes a model for a specificand written in SFOAL.

Models.

Intuitively a model for a SFOAL specification will contain a collection of scenarios and a collection of scenario transformation functions; the idea is that the functions represent the actions of a specificand and model movement between the scenarios in accordance with the descriptions of actions. As we shall see there may be many such models for any given specificand. We will begin with some preliminary definitions.

Definition: Theory Extension.

To state the relationship between a specificand and a scenario we need to introduce the notion of a SFOAL theory extension. Given two SFOAL theories, s and s' , over the same language L , the latter is an extension of the former, written $s \sqsubseteq s'$, iff

$$\forall \varphi \text{ if } \vdash_s \varphi \text{ then } \vdash_{s'} \varphi.$$

In other words one theory is an extension of another if and only if the former proves at least the same number of properties as the latter in a common language.

Definition: Consistent Scenario.

A scenario s is consistent if and only if it is not the case for any formula ϕ that both $\vdash_s \phi$ and $\vdash_s \neg \phi$ hold.

We shall distinguish specificands from scenarios by writing $\mathbf{sp}$ to refer to the theory representative of a particular specificand. A scenario is then a theory extension of the theory $\mathbf{sp}$. Remember a specification is a special SFOAL theory - one that is closed under the rule $\mathcal{R}_2$.

Model Structures.

A *model structure* for a SFOAL specificand $\mathbf{sp}$ is given by a pair $\langle \mathcal{S}, \mathcal{F} \rangle$ where

$\mathcal{S}$: is a non empty set of potential scenarios, where a scenario, as we defined earlier, is a theory extension of the specificand $\mathbf{sp}$ and hence

$$\mathcal{S} \subseteq \{ s : \mathbf{sp} \sqsubseteq s \text{ \& } s \text{ is consistent} \}$$

Note that by the definition of theory extension above, the theory corresponding to the specificand may also be included in the set $\mathcal{S}$.

$\mathcal{F}$: is a collection of typed (total) functions, one for each action name in the syntactic category $\mathbf{Act}$ of the specification $\mathbf{sp}$. The type of such an action is

$$f : t(\alpha) \times \mathcal{S} \rightarrow \mathcal{S}$$

where $t(\alpha)$ denotes the type of the action name α , and $\mathcal{S}$ is as above. Note that action constants will be represented by functions from scenarios to scenarios. The definition of a model is given overleaf and is expressed as a constraint on model structures.

Models.

A model structure $\langle \mathcal{S}, \mathcal{I} \rangle$ for $\mathbf{sp}$ is a model for $\mathbf{sp}$ iff,

- $\forall s \in \mathcal{S} \quad \forall \alpha \in \mathbf{Act}$
 $\text{if } [\alpha] \varphi \in \mathbf{TH}(s) \text{ then } \varphi \in \mathbf{TH}(f_\alpha(s))$

Where f_α represents the function corresponding to the action name α , and $f_\alpha(s)$ denotes the scenario resulting from applying the function f_α to the scenario s . This condition is intended to reflect the informal meta rule $\mathcal{M}1$. As above, from here on we shall use the set membership symbol $\in$ (and its contrapositive $\notin$) as an informal but conventional meta device to indicate whether or not a formula belongs to a particular theory.

- $\forall s \in \mathcal{S} \quad \forall \alpha \in \mathbf{Act}$
 $\text{if } \neg [\alpha] \varphi \in \mathbf{TH}(s) \text{ then } \varphi \notin \mathbf{TH}(f_\alpha(s))$

This condition is similar to the previous one and reflects the informal meta rule $\mathcal{M}2$.

As a consequence of having implicit and incomplete states of information this definition of model is very loose indeed. According to our definition a specification will admit a very large number of models - some of which may not be intuitive. As an illustration of this consider the following specification:

$$\text{Connected}(n, n') \rightarrow [\text{Send}(m, n, n')] \text{At}(m, n').$$

This is to be read: "If node n is connected to node n' then sending a message m from n to n' will result in the message being at the destination node". Now take $\mathcal{S}$ to consist of a single scenario in which the following properties hold:

$$\neg \text{Connected}(n, n') \quad \text{and} \quad \neg \text{At}(m, n')$$

then, this by the above definition is a model for the specification above. The reason for this is that the above axiom acts like a counterfactual, it does not demand that there should ever be a situation in which node n is connected to node n' , simply that if it does and a message is sent then it is considered to be at the destination node as a result.

At first this may seem to suggest that the functions modelling actions should be partial - this is not so. In order to see that functions in our models are total we need look no further than the model condition corresponding to $\mathcal{M}1$; this says that if in any scenario

we have an expression of the form $[\alpha] \phi$ then applying the function corresponding to α to this scenario will result in a scenario in which it is possible to prove that ϕ holds. But, for any scenario of a specification we can always show that for all actions a description of the form $[\alpha] \phi$ holds: in the case of a non empty specification this follows from the necessitation rule and in the case of an empty specification it follows from the derived axiom $[\alpha] \text{TRUE}$. Hence if $\mathcal{M} \models$ is to be observed then the application of any function corresponding to an action must result in some object from which some property can be proven and since the co-domain of these functions are scenarios this object must be a scenario.

Hence, in the example above the function corresponding to the send message action would have to be represented (considering that only one scenario exists) as the identity function. Another model for the above specification may consist of two scenarios, both of which have the property that nodes n and n' are not connected but distinguished by whether or not the message m is at node n' . But even in this case the function corresponding to the above action can still be defined as the identity function on scenarios. In general this shows that given a fixed interpretation for some collection of functions there may still be many different collections of scenarios which may be used to construct a model for a specification.

Conversely, given a fixed set of scenarios there are in general many different collections of functions which satisfy the above model constraints. This was seen as a direct consequence of not fully determining the effects of actions. Two kinds of underdeterminedness were identified, implicit and explicit. The implicit kind of underdeterminedness is due to the fact that action descriptions are only indicative of the scenario reached by performing an action. Thus although action descriptions of the form $[\alpha] \phi$ describe an action that results in a scenario in which the property ϕ holds, it says nothing about how the resultant scenario relates to the scenario in which α was performed.

Some researchers have proposed to augment their formalisms with something called a frame axiom which states that the scenario in which an action is performed is *most like* the scenario resulting from performing the action whilst being consistent with the explicit consequence of that action. The use of the phrase "most like" is to stipulate minimum change between scenarios, consistent with the action being performed. However as we saw above there may not be a unique notion of minimal change and so implicit

underdeterminedness still exists. Explicit underdeterminedness also leads to a choice of resultant scenarios when an action is performed, the most clear example of this is when the explicit consequence of an action contains a disjunction.

Both the above kinds of underdeterminedness give rise to a choice of scenarios when any action is performed. In most traditional modal logic settings underdeterminedness is usually handled by defining a relation between scenarios (possible worlds). This relation is usually called an accessibility or reachability relation. If we had chosen this approach then a model for a SFOAL specification would simply consist of a collection of scenarios $\mathcal{S}$ and a relation $\mathcal{R}$ of triples of type $\langle \mathcal{S}, \text{Act}, \mathcal{S} \rangle$ such that

$$\langle s, \alpha, s' \rangle \in \mathcal{R} \quad \text{iff} \quad \vdash_s [\alpha] \varphi \quad \text{and} \quad \vdash_{s'} \varphi.$$

This makes the choices involved in deciding upon the scenario resulting from performing an action quite an explicit part of the model. In contrast to this we have chosen to deal with underdeterminedness in a more implicit fashion by requiring that action be modelled by scenario transforming functions; this results in a choice of functions for any particular action. Further since a model is defined in terms of a collection of functions, one per action, this gives rise to a number of different models, one model for each particular collection of functions.

The decision to adopt a functional account rather than a relational one was based on the intuition that the former seems to correspond more naturally to the relationship between a specification and potential implementations. In general a specification has many implementations just as in our formalism a specification has many models. However we do not in general assume that because a specification is loose that the corresponding implementation will be nondeterministic, which corresponds more to a relational setting, but rather that there are perhaps many different acceptable functions. Hence we must clearly distinguish between underdeterminedness and true nondeterminism. In order to model truly nondeterministic actions a relational account is essential; we shall see this when we consider the nondeterministic action combinator.

It is important to note that this approach to actions being modelled by functions is a model level decision and is not a property of our logic; the logic is incomplete with respect to this property. The reason for this is that we do not force scenarios to be complete theories; if we did - essentially by including the axiom $\neg [\alpha] \varphi \rightarrow [\alpha] \neg \varphi$ - then we could prove that all scenarios were complete information scenarios.

There are several reasons why we have chosen to represent scenarios as, in general, incomplete theories. The most important being that it seems, to us at least, more appropriate to represent information about states (as opposed to the states themselves) by theories. The justification for this is more or less one of taste, although there is something in the claim that theories, and in particular the notion of proof within a theory, provide a more amenable mechanism for conveying information states. For example, the notion of proof provides a conceptual way of "computing" the information in a scenario. Another reason for choosing theories instead of truth functional structures (we avoid the more conventional term model as we have used it differently above) is that it is easier to handle incomplete information. This is essentially due to the fact that for certain formulae it may be the case that neither they nor their negations are provable from a given scenario. If this is the case we have incomplete information with respect to these formulae. In this sense it would not be incorrect to see a scenario as a representative of a collection of more informative (complete information) system states. System scenarios could then be made more and more informative by refining and enriching the corresponding specification to include not only requirements but also detailed design decisions. In an ideal world we could view this process of refinement as leading from requirement specification to system implementation and correspondingly from abstract system scenarios to actual system states.

In short the decision to represent scenarios as SFOAL theories reinforces our view that a requirement specification should be seen as defining an information structure *about* a system rather than as defining an implementation *of* a system.

Another point worth mentioning about specifications and their models concerns the idea of reachability; in general, this term is used to classify all those states/scenarios which can be reached by performing actions. In the above definition of models this is dependent on two factors, the concept of an initial scenario and on the definition of the functions in that model. A scenario is said to be reachable from some other scenario s , if some function (corresponding to an action) when applied to s results in that scenario. Given an initial scenario the notion of a reachable model can then be defined as all scenarios reachable by the functions (including composition of such functions) of a particular model. The point to note is that a specific model for a specification in general contains many reachable models. This is due to the fact that although the scenario morphisms associated with any model are total they are not onto. The fact that a model as we have defined it above may have many reachable models is particularly satisfying

when we consider that requirement specifications may not in general force a particular initial scenario. This could be left either as an implementation or design decision. A particular implementation would then choose one reachable model out of the many provided by the specification and a particular design decision (perhaps achieved by refining the existing requirement specification) might narrow down the choice between reachable models.

Lastly, a few words about the inclusion of the derived axiom, axiom six, are in order - the form of which is $[\alpha] \text{TRUE}$. To see how this is an implicit consequence of any specification all we need do is observe that TRUE is a property of any specification and hence (by the necessitation rule) so is $[\alpha] \text{TRUE}$ for any action α . Hence, although axiom six can be derived from the necessitation rule it is a bit of a special case which deserves special attention. There are two reasons for this. Firstly it was incorporated into our logic of actions to highlight the intuitive concept of the least determined action description and, secondly, (and perhaps more technically) because it is the nearest we can come to capturing in axiom form the intent of the necessitation rule. Remember that the necessitation rule applies to any property of the specification and hence is specification specific and so cannot be axiomatised. Truth on the other hand is a consequence (by construction/definition) of all specifications and so can be talked about directly in the axiomatics of our logic.

In the next section we will talk about the role of specific SFOAL formulae as the means for expressing system requirements in the context of a more concrete example.

2.5 An Example.

In the previous section we defined a requirement specificand to be a special kind of SFOAL theory. Although we gave abstract justifications for certain choices and concepts we have not yet provided any concrete illustration of what such a specificand would look like. In this section we would like to rectify this by presenting and discussing a more complete example. As well as providing an illustration of how SFOAL expressions can reflect certain "real" world phenomena, it will also serve to show the kind of specificand and scenario based reasoning we can perform. The structure of the section is as follows: first we shall present the example specificand and discuss what it "says" (that is what requirements it captures), then we shall see what kind of reasoning (based on this specificand) our formalism will allow and, finally, we shall conclude with some comments about the refinement of requirement specifications.

The example we have chosen involves capturing the requirements of a small shipping line, the name of which is Atarman, and consists of two ships with licences to trade at three ports. The business of the company is to carry three kinds of cargo between the ports it is licenced for. However, in order to retain its licence to trade the company must obey certain South American trading laws. Hence, to remain within the law and retain their trading licences the company has commissioned our services in drawing up a document detailing all the legal requirements it must observe. The resulting document is to be used as a starting point for designing their shipping schedules. After spending vast amounts of time deciphering legal gobbledegook the commissioned specifiers arrived at the concise statement below. They have of course chosen SFOAL as their medium of expression.

Specification Atarman.

Language

Sorts: Ships, Ports, Cargo, Nat, Act.

Constants for:

Ships	Voyager, Discovery.
Ports	Boston, Dover, Bombay.
Cargo	Tea, Steel, Timber.

Variables for:

Ships	s, s', s'',
Ports	p, p', p'',
Cargo	c, c', c'',

Function symbols:

Weight	of type < Ships, Cargo, Nat >
Atport	of type < Ships, Port >
Total	of type < Ships, Nat >

Predicate symbols:

Carries	of type < Ships, Cargo >
= _s	of type < Ships, Ships >
= _p	of type < Ports, Ports >
= _c	of type < Cargo, Cargo >

Action names:

Sail	of type < Ships, Port, Act >
Unload	of type < Ships, Cargo, Act >

Axioms.

- 1) $\forall s \text{ Weight}(s, \text{Timber}) \leq 50$
- 2) $\forall s \forall c \text{ Weight}(s, c) = 0 \rightarrow \neg \text{Carries}(s, c)$
- 3) $\forall s \forall c \text{ Weight}(s, c) > 0 \rightarrow \text{Carries}(s, c)$
- 4) $\forall s \forall c \text{ Total}(s) = \text{Weight}(s, \text{Tea}) + \text{Weight}(s, \text{Steel}) + \text{Weight}(s, \text{Timber})$
- 5) $\forall s \text{ Total}(s) \leq 200$
- 6) $\forall s \forall p [\text{Sail}(s, p)] \text{ Atport}(s) = p$
- 7) $\forall s \forall c [\text{Unload}(s, c)] \neg \text{Carries}(s, c)$
- 8) $\forall s \text{ Atport}(s) = \text{Bombay} \rightarrow [\text{Sail}(s, \text{Boston})] \text{ Weight}(s, \text{Tea}) \geq 50$
- 9) $\text{Atport}(\text{Voyager}) = \text{Boston}$
 $\rightarrow [\text{Sail}(\text{Voyager}, \text{Dover})] \text{ Weight}(\text{Voyager}, \text{Steel}) \leq 100$
- 10) $\forall s [\text{Sail}(s, \text{Dover})][\text{Unload}(s, \text{Steel})][\text{Sail}(s, \text{Boston})] \text{ Weight}(s, \text{Timber}) = 50$

Before we go on to see exactly what the above specification says, a few notational points are in order. So as not to (unnecessarily) clutter the specificand we have written equivalence without the appropriate subscript and in infix style, the appropriate equality being determined by context. We have also assumed the availability of the natural numbers (Nat) with various operations (not given here, but standard) such as addition (+), less than or equal to ($\leq$), etc. Giving these details would just obscure the requirements of our shipping company and not add any further insights to our formalism. We are now in a position to examine the above specificand.

We shall look first at the language; most of this is self explanatory, for example the company has two ships called Voyager and Discovery and there are three ports at which it is allowed to trade: Boston, Dover and Bombay. Further there are two actions which the specificand mentions, the action of a ship sailing to a particular port (Sail) and the action of a ship unloading a particular cargo (Unload). A number of functions are also defined: the function Weight reflects the amount of tonnage of a particular cargo that a ship is carrying, whereas the function Atport reflects the port that a ship is at and Total is supposed to keep track of the total tonnage a ship is carrying. The predicate Carries is a relationship between ships and kinds of cargo and signifies when a ship is carrying a particular kind of cargo. Note that tonnage is indicated by natural number values returned by the functions Total and Weight.

Having given the general intuition behind the language of our specification we can now see what the specificand "says" by interpreting the meaning of the above axioms. We can split this collection of formulae into two kinds: those which talk about static requirements (those which are invariant to all change) and dynamic requirements (those which talk about more specific constraints on change). The first five axioms are of the first kind whereas the last five are of the second; we shall examine each type in turn.

As we have already said the first five axioms are intended to be constraints invariant with respect to change (we shall show later why this is when we come to reason about the specification); for the present we shall concern ourselves only with what they mean. Axiom one restricts all ships to carry cargoes of timber not greater than fifty tons. The next two axioms, two and three, define the characteristic function of the predicate Carries. Taken together they say that a ship carries a particular type of cargo if it is carrying a non zero amount of that particular cargo kind and does not carry that kind of cargo if it is carrying a null amount of it. The next axiom, axiom four, defines what we

mean by the total tonnage of a ship in terms of the amount of each cargo it is carrying. The definition simply involves equating the additive amounts of all three kinds of cargo a ship is carrying to the value of the Total function. Finally the fifth axiom constrains all ships to carrying a maximum overall tonnage of two hundred tons.

The next five axioms deal with the more specific (local) dynamic requirements concerned with the operational logistics of the shipping company. The first two of these, axioms six and seven, simply define the results of performing the actions Sail and Unload. The first says that after a ship sails to a particular port then that ship is said to be at that port, and the second says that whenever any ship unloads a particular kind of cargo the result is indicated by the fact that that ship no longer carries that particular cargo. We turn now to the last three axioms which are perhaps also the most interesting.

Axiom eight is an example of a conditional modality: its purpose is to relativise (or condition) the description of an action; its general form is

$$\psi \rightarrow [\alpha] \phi$$

The intuitive reading of this is: "If ψ holds in a scenario, then performing the action α in that scenario will result in a scenario in which ϕ holds". Further, if a conditional modal expression is part of a specification then using the static interpretation of the necessitation rule we know that the expression " $\psi \rightarrow [\alpha] \phi$ " says that *any* scenario in which ψ holds is a scenario in which $[\alpha] \phi$ holds (that is, is a scenario in which the action α being performed results in a scenario in which ϕ holds). In general the effect of an action may vary depending upon the context in which it is performed and the mechanism of material implication provides a convenient way of representing this. Take the following example:

$$\phi \rightarrow [\alpha] \psi$$

$$\neg \phi \rightarrow [\alpha] \tau.$$

This says that when ϕ holds, performing α results in ψ holding, whereas when $\neg \phi$ holds then the effect of performing α is τ . At first this may seem as though the action α is forced to either result in ψ or τ whenever it is performed but we must remember that there may be scenarios in which neither ϕ nor its negation hold and consequently neither may $[\alpha] \psi$ or $[\alpha] \tau$.

Returning now to the specific context of the above example we can see that axiom eight

requires that any ship travelling from Bombay to Boston must arrive in Boston with a Cargo of at least fifty tons of tea. This in turn means that it must have left Bombay with a consignment of at least fifty tons of tea. Axiom nine is another example of a conditional action description, except that it is an even more specific constraint than the previous one. It should be read as saying that whenever the ship named Voyager sets sail from Boston and is bound for Dover it must be carrying at most one hundred tons of steel.

The last axiom also deserves special discussion, it is an example of an iterated modality, or an action description which is relative to the performance of some other action. For the purposes of illustration consider the expression

$$[\alpha][\beta] \varphi$$

where φ is a non modal expression; this is to be understood as "after performing the action α the resulting scenario, s' say, will exhibit the modal property $[\beta] \varphi$. In other words performing the action β in s' will result in a scenario in which φ holds". This ability of stating action descriptions as the effects of other actions is not available in specification languages such as Z or VDM, which adopt a Hoare-like style for the description of actions. In both these formalisms the only mechanism for describing actions is based on the pre/post-condition idea where if a certain property (pre-condition) holds before an action is executed, then after that action's execution some other property (the post-condition) holds. The post condition in either language is not allowed to be another action description. It can be argued that iterated modalities can be implicitly coded into VDM or Z by defining unique chaining properties and using meta level reasoning facilities. To make this point clear assume the following action descriptions written in VDM, where we omit precise VDM syntax and only look at the form of the descriptions

Pre: Null.	Action: α .	Post: a .	No side effects.
Pre: a .	Action: β .	Post: $\varphi \wedge \neg a$.	No side effects.

Given that the property a can only be made to come about by action α and can only be switched off by action β , then using the compositional reasoning of VDM we can conclude that " whenever an execution of action β is preceded by an occurrence of action α then the property φ holds". There are two major reasons why this coding is not acceptable; firstly, we are forced to move away from simple action description and apply some reasoning mechanism to arrive at the desired conclusion, which makes for an

implicit treatment. And secondly, the introduction of secondary properties whose sole role is to allow the above kind of reasoning is both unnatural and can lead to unnecessarily cluttered specifications. Since clarity and readability are both factors of prime concern as far as specifications are concerned, the above codification is unacceptable. The above comments apply equally to Z; the reader should note that the clauses stating that no side effects are allowed are necessary for the above reasoning process to be correct.

Having identified a feature of our formalism not available in Z or VDM we should perhaps say something about its use. Perhaps the most important advantage of allowing action descriptions to appear as the "effects" of other actions is that we can broaden our notion of a context to include not only first order properties but also information about the performance of actions as well. The modal expression $[\alpha][\beta] \phi$ can in a pre/post condition style be rewritten as:

Pre: action α . Action: β . Post: ϕ .

Here, the precondition states that α is the last action which is performed. This means that we can now formulate action descriptions relative to information about which action(s) have just been performed. Natural language system specifications often contain descriptions of this kind, for example: "If the reactor is shut down following a maintenance routine then an emergency alert condition ensues", that is $[\text{maintenance}][\text{shut}] \text{ alert}$.

It is easy now to see what axiom ten is telling us; we can sum it up as the following requirement: "Whenever a ship sails to Dover (no matter from whence it came to Dover) and after unloading its steel there sails to Boston, it must arrive at Boston with the maximum allowed consignment of timber cargo (fifty tons, see axiom one)".

Looking back over the specification it seems as though modal expressions play a dual role, they act both as a means of describing actions and as a mechanism for expressing dynamic constraints. It could, for instance, be said that axiom six is an example of an action description and axiom ten is an example of a dynamic constraint. Our own view is that both axioms give some information about the actions they mention and it is a matter of taste whether or not one regards either one a description or a constraint. In the case of axiom ten, we could think in terms of the axiom providing us with more information about the action of sailing (to Boston) rather than as a constraint. It is perhaps more

constructive to see things this way as it helps in the task of implementing the action of (legally) sailing ships between ports.

An example of where the distinction between viewing modal expressions as constraints or action descriptions is most interesting is when the only information we have about a certain action is relative to the occurrence of some other action. For example, suppose we wanted to define the action of loading cargo and at this particular time all we know is that any ship sailing from Dover and loading must carry tea to its next destination. We could capture this information as follows:

$$\forall s \forall c \text{ [Sail}(s, \text{Dover})][\text{Load}(s)] \text{ Carries}(s, \text{Tea}).$$

Note that this does not commit us to any particular amount of tea except of course a non zero amount. Would we say that the above was an expression that described the action of loading or is it a constraint on the action of loading ? As we have already said, the answer is a question of taste, we ourselves preferring to see it as partially describing the action load. There are of course many other interesting SFOAL expressions we could discuss; unfortunately we do not have space here for all of them. We hope however that the above example and the ensuing discussion has given the reader some taste for the expressive power of the language.

So far we have focused mainly on the explicit representation of information, but an important aspect of any requirement specification formalism is the kind of reasoning it allows one to perform. In light of the previous section, we can identify two kinds of reasoning that we can perform on specifications written in SFOAL, reasoning about the specification itself and reasoning about a scenario of a specification. We shall deal with reasoning from a specification first.

In the above example of a shipping company, although we only looked at the meaning of the explicit axioms, there are many implicit consequences of the specification which we did not touch on. For example, although not explicitly stated, we can prove that any ship which sails to Dover, unloads her cargo of steel there and then sails on to Boston must carry a combined cargo of Tea and Steel of no more than one hundred and fifty tons. This in the symbolism of SFOAL is

$$\begin{aligned} & [\text{Sail}(s, \text{Dover})] ([\text{Unload}(s, \text{Steel})] [\text{Sail}(s, \text{Boston})] \\ & (\text{Weight}(s, \text{Tea}) + \text{Weight}(s, \text{Steel}) \leq 150)). \end{aligned}$$

The proof that this is a consequence of our specification is as follows; in the interests of clarity we shall avoid the explicit universal quantification of variables and take this to be implicit.

Proof:

1. $\text{Total}(s) = \text{Weight}(s, \text{Tea}) + \text{Weight}(s, \text{Steel}) + \text{Weight}(s, \text{Timber})$ A1
2. $\text{Total}(s) \leq 200$ A2
3. $\text{EQ} \wedge \text{Total}(s) \leq 200$ FOL. 1,2
4. $[\text{Sail}(s, \text{Boston})] \text{EQ} \wedge \text{Total}(s) \leq 200$ Ness. 3
5. $[\text{Unload}(s, \text{Steel})][\text{Sail}(s, \text{Boston})] \text{EQ} \wedge \text{Total}(s) \leq 200$ Ness. 4
6. $[\text{Sail}(s, \text{Dover})]$
 $([\text{Unload}(s, \text{Steel})][\text{Sail}(s, \text{Boston})] (\text{EQ} \wedge \text{Total}(s) \leq 200))$ Ness. 5
7. $[\text{Sail}(s, \text{Dover})]$
 $([\text{Unload}(s, \text{Steel})][\text{Sail}(s, \text{Boston})] (\text{Weight}(s, \text{Timber}) = 50))$ A10
8. $[\text{Sail}(s, \text{Dover})] ([\text{Unload}(s, \text{Steel})][\text{Sail}(s, \text{Boston})]$
 $(\text{Weight}(s, \text{Timber}) = 50) \wedge (\text{EQ} \wedge \text{Total}(s) \leq 200))$ SFOAL(11) 6,7
9. $[\text{Sail}(s, \text{Dover})] ([\text{Unload}(s, \text{Steel})] [\text{Sail}(s, \text{Boston})]$
 $(\text{Total}(s) = \text{Weight}(s, \text{Tea}) + \text{Weight}(s, \text{Steel}) + 50) \wedge (\text{Total}(s) \leq 200))$
[50/Weight(s, timber)] in EQ.
10. $[\text{Sail}(s, \text{Dover})] ([\text{Unload}(s, \text{Steel})] [\text{Sail}(s, \text{Boston})]$
 $((\text{Weight}(s, \text{Tea}) + \text{Weight}(s, \text{Steel}) + 50) \leq 200))$ FOL 9
11. $[\text{Sail}(s, \text{Dover})] ([\text{Unload}(s, \text{Steel})] [\text{Sail}(s, \text{Boston})]$
 $(\text{Weight}(s, \text{Tea}) + \text{Weight}(s, \text{Steel}) \leq 150))$

Q.E.D

The first two lines (which form the initial assumptions in the above proof) are explicitly

stated as part of our specification, the third line is simply the conjunction of these two properties. Lines four, five and six are obtained by repeated application of the necessitation rule, axiom seven being an explicit axiom of our specification. The next step rests on the property that

$$([\alpha] \phi \wedge [\alpha] \psi) \leftrightarrow [\alpha] (\phi \wedge \psi)$$

is an axiom of SFOAL. Line nine is simply line eight except with the substitution of the number 50 in place of the term Weight(s, timber): we are allowed this because of the explicit equality between 50 and Weight(s, Timber) in the previous line and the laws of substitution in first order logic. Line ten rewrites the expression, Total(s) $\leq$ 200, using the equivalence between Total and the summation of the individual weights of cargo type. Finally line eleven is arrived at by a simple manipulation of the final inequation.

The example above was perhaps trivial but it does go to show how reasoning from a specification *can* be carried out. In more complex specifications the proof (if it exists) of certain properties may not be so simple. This process of seeing whether or not certain system properties can be deduced from the specification is also a good way for the people who have written the specification to gain a certain amount of confidence in it. The specifiers may, for example, ask the clients who commissioned the specification whether the system they require conforms to certain implicit constraints which they have derived. If not, they know that something is wrong with the existing specification and can take steps to put things right. This confidence building process also works in reverse; the people commissioning the specification may request the authors to prove to them that certain essential properties can be shown to hold of the specification (and hence of all potential implementations).

Reasoning about a potential system in terms of its specification (in the above manner) is all very well, but there may be times when we wish to be more specific in the kind of reasoning we want to perform. Scenario based reasoning is an example of this; the purpose of such reasoning is to help "validate" a requirement specification. We do not (here) mean validation in a formal correctness sense; this would be pointless since we would need something against which we could prove correctness, and this is presumably the very object we are trying to formulate in the first place. By validation we mean a kind of intelligent "testing" of the specificand. Instead of simply reasoning with a specificand and making sure it exhibits certain desirable properties we may wish to test it by postulating (augmenting it with) certain other facts corresponding to envisaged

system situations, and seeing what consequences ensue. Again, returning to our simple-minded example above, suppose we wished to see the consequences of being in a scenario in which the ship named Voyager was docked at Bombay and the situation was such that we also knew that if it sailed to Boston it would be carrying the maximum allowed cargo of timber, fifty tons. We can represent these two more specific pieces of information as follows:

* $\text{Atport}(\text{Voyager}) = \text{Bombay}$

** $[\text{Sail}(\text{Voyager}, \text{Boston})] (\text{Weight}(\text{Voyager}, \text{Tea}) \geq 50 \rightarrow$
 $[\text{Sail}(\text{Voyager}, \text{Dover})] \text{Weight}(\text{Voyager}, \text{Timber}) = 50) .$

The first of these constitutes a simple fact about the present (hypothetically depicted) scenario whilst the second states an action description specific to the present scenario, and so provides information (in this case another action description) about another potential scenario. Note that the action description, as is the simple property, is specific to the scenario about which we wish to reason and is therefore not the same as action descriptions which are part of a specification. More specifically the necessitation rule we cited earlier is not applicable to either (scenario specific) property; remember that the necessitation rule is applicable to only properties of the specification. The reader is referred back to an earlier point in this chapter where the reasons for this are given. Having set up this imaginary scenario, about which we know the two properties * and **, it is time we demonstrated how scenario based reasoning may be carried out. Although our (toy) example will be short it will be sufficient to demonstrate the purpose and mechanics of scenario (as opposed to specification) based reasoning.

For the purposes of our example we shall assume that all reasoning (proofs) are performed relative to a scenario of the shipping company Atarman, consisting of the two properties * and **. In other words our reasoning is relative to the theory

$$\text{TH}(\text{TH}(\text{Atarman}) \cup \{ *, ** \})$$

where $\text{TH}(\text{Atarman})$ represents the special (closed under the necessitation rule) SFOAL theory (collection of provable formulae) corresponding to the specification Atarman given earlier. The expression $\text{TH}(\text{TH}(\text{Atarman}) \cup \{ *, ** \})$ then represents the collection of formulae provable by augmenting the set of formulae $\text{TH}(\text{Atarman})$ by the two properties * and **. The specific theory presentation of $\text{TH}(\text{TH}(\text{Atarman}) \cup \{ *, ** \})$ we shall use will be the union of the theory presentation (given earlier) of

TH(Atarman) and the two new properties * and **. Note that the emboldened theory expansion operator **TH** does not incorporate the necessitation rule. For convenience we shall now list this out in full.

- (1) $\forall s \text{ Weight}(s, \text{Timber}) \leq 50$
- (2) $\forall s \forall c \text{ Weight}(s, c) = 0 \rightarrow \neg \text{Carries}(s, c)$
- (3) $\forall s \forall c \text{ Weight}(s, c) > 0 \rightarrow \text{Carries}(s, c)$
- (4) $\forall s \forall c \text{ Total}(s) = \text{Weight}(s, \text{Tea}) + \text{Weight}(s, \text{Steel}) + \text{Weight}(s, \text{Timber})$
- (5) $\forall s \text{ Total}(s) \leq 200$
- (6) $\forall s \forall p [\text{Sail}(s, p)] \text{ Atport}(s) = p$
- (7) $\forall s \forall c [\text{Unload}(s, c)] \neg \text{Carries}(s, c)$
- (8) $\forall s \text{ Atport}(s) = \text{Bombay} \rightarrow [\text{Sail}(s, \text{Boston})] \text{ Weight}(s, \text{Tea}) \geq 50$
- (9) $\text{Atport}(\text{Voyager}) = \text{Boston}$
 $\rightarrow [\text{Sail}(\text{Voyager}, \text{Dover})] \text{ Weight}(\text{Voyager}, \text{Steel}) \leq 100$
- (10) $\forall s [\text{Sail}(s, \text{Dover})][\text{Unload}(s, \text{Steel})]$
 $[\text{Sail}(s, \text{Boston})] \text{ Weight}(s, \text{Timber}) = 50$
- (11) $\text{Atport}(\text{Voyager}) = \text{Bombay}$
- (12) $[\text{Sail}(\text{Voyager}, \text{Boston})] (\text{Weight}(\text{Voyager}, \text{Tea}) \geq 50$
 $\rightarrow [\text{Sail}(\text{Voyager}, \text{Dover})] \text{ Weight}(\text{Voyager}, \text{Timber}) = 50) .$

We shall now proceed to prove that the following property

$$[\text{Sail}(\text{Voyager}, \text{Boston})] ([\text{Sail}(\text{Voyager}, \text{Dover})]$$

$$\text{Weight}(s, \text{Tea}) + \text{Weight}(s, \text{Steel}) \leq 150).$$

which is not directly obvious, holds of this scenario.

Proof:

1. $\text{Atport}(\text{Voyager}) = \text{Bombay}$ A11.
2. $\forall s \text{ Atport}(s) = \text{Bombay}$
 $\rightarrow ([\text{Sail}(s, \text{Boston})] \text{Weight}(s, \text{Tea}) \geq 50)$ A8.
3. $\text{Atport}(\text{Voyager}) = \text{Bombay}$
 $\rightarrow [\text{Sail}(\text{Voyager}, \text{Boston})] \text{Weight}(s, \text{Tea}) \geq 50$ [Voyager/s] in 2.
4. $[\text{Sail}(\text{Voyager}, \text{Boston})] \text{Weight}(\text{Voyager}, \text{Tea}) \geq 50$ M.P. 1,3.
5. $[\text{Sail}(\text{Voyager}, \text{Boston})] (\text{Weight}(\text{Voyager}, \text{Tea}) \geq 50$
 $\rightarrow [\text{Sail}(\text{Voyager}, \text{Dover})] \text{Weight}(\text{Voyager}, \text{Timber}) = 50)$ A12.
6. $([\text{Sail}(\text{Voyager}, \text{Boston})] \text{Weight}(\text{Voyager}, \text{Tea}) \geq 50)$
 $\rightarrow ([\text{Sail}(\text{Voyager}, \text{Boston})]$
 $[\text{Sail}(\text{Voyager}, \text{Dover})] (\text{Weight}(\text{Voyager}, \text{Timber}) = 50)$ SFOAL(7). 5 .
7. $[\text{Sail}(\text{Voyager}, \text{Boston})]$
 $([\text{Sail}(\text{Voyager}, \text{Dover})] \text{Weight}(\text{Voyager}, \text{Timber}) = 50)$ M.P. 4,5
8. $\text{Total}(s) = (\text{Weight}(s, \text{Tea}) + \text{Weight}(s, \text{Steel}) + \text{Weight}(s, \text{Timber}))$ A4
9. $\text{Total}(s) \leq 200$ A5
10. $\text{EQ} \wedge \text{Total}(s) \leq 200$ FOL. 8,9
11. $[\text{Sail}(\text{Voyager}, \text{Dover})] \text{EQ} \wedge \text{Total}(s) \leq 200$ Ness. 10
12. $[\text{Sail}(\text{Voyager}, \text{Boston})]$
 $([\text{Sail}(\text{Voyager}, \text{Dover})] \text{EQ} \wedge \text{Total}(s) \leq 200)$ Ness. 11

13. $[Sail(Voyager, Boston)]$
 $([Sail(Voyager, Dover)] ((Weight(Voyager, Timber) = 50)$
 $\wedge (EQ \wedge Total(s) \leq 200)))$ SFOAL(11). 7,12.
14. $[Sail(Voyager, Boston)] ([Sail(Voyager, Dover)]$
 $(Total(s) = Weight(s, Tea) + Weight(s, Steel) + 50$ [50/Weight(s, timber)]
 $\wedge (Total(s) \leq 200)))$ in EQ.13.
15. $[Sail(Voyager, Boston)] ([Sail(Voyager, Dover)]$
 $(Weight(s, Tea) + Weight(s, Steel) + 50) \leq 200))$ FOL. 14.
16. $[Sail(Voyager, Boston)] ([Sail(Voyager, Dover)]$
 $(Weight(s, Tea) + Weight(s, Steel) \leq 150))$ Q.E.D.

Although quite lengthy, the above proof is remarkably simple; its purpose is to prove that whenever the ship Voyager sails to Boston and then on to Dover its combined cargo of tea and steel is not more than one hundred and fifty tons. From axiom eight of the specification we know that all ships which sail to Boston from Bombay arrive there with a cargo of tea not less than fifty tons; this is specifically true for the ship Voyager which is at Bombay in the particular scenario we are dealing with. This, together with another property of this scenario - that is, whenever the Voyager sails to Boston and arrives with a cargo of not less than fifty tons of tea and then sails on to Dover means that it must arrive in Dover with fifty tons of timber - allows us to conclude that if the Voyager sailed first to Boston and then to Dover (from its present port) then its cargo of timber upon arrival at Dover must be fifty tons. In addition, since we also have the scenario invariant constraint that all ships must not exceed a total cargo tonnage of two hundred this means that the remaining cargo of the Voyager, tea and steel, must not exceed one hundred and fifty tons.

An important feature of this proof is the introduction of action descriptions specific to a particular scenario; in general this is a very powerful tool for reasoning (hypothetically) about how a system will react in a certain situation. Broadly speaking, the introduction of action descriptions as scenario specific assumptions allows us to reason about how a system will react given that certain actions (with specific consequences) are performed.

This provides us with an extra dimension with which to experiment with in the validation or *intelligent* testing activities we mentioned earlier. Applications which require either prediction or hypothetical simulation would be especially applicable to this form of reasoning.

2.6 Conclusions.

Having devoted this chapter to discussing the virtues of SFOAL as a formalism for stating system requirements it will be appropriate for us to end with a major shortcoming: the description/modelling of a system's behaviour. If we consider the shipping example above we see that although the dynamics of ship movement/activity were described by the actions of sailing between ports and unloading cargo, we left unstated information about which ports a ship *could* sail between and when it is allowed to unload cargo. Take, for example, the following expression:

$$\forall s \forall p \text{ [Sail}(s, p)\text{]} \text{ Atport}(s) = p$$

As we saw earlier this states that if any ship *s* sails to a port *p* then on completion of that voyage the ship is said to be at port *p*. The operative phrase here is the hidden conditional: "if any ship sails"; we believe that such statements need to be distinguished from statements which express when a ship is allowed to sail to various ports. The former should be used to (usually incompletely) describe all possible scenario transformations whereas the latter should be used to describe the behaviour of a system, essentially by pruning this transition space so that certain transitions are disallowed. There are many ways in which this can be achieved, perhaps the best known is the use of preconditions. Hence an expression such as

$$\text{I} \quad \forall s \forall p \neg (\text{Atport}(s) = p) \rightarrow ([\text{Sail}(s, p)] \text{Atport}(s) = p)$$

can be seen (as long as our notion of model is changed to admit scenario transforming functions which are partial) as restricting the description of the sailing action to ships which are not already at the ports to which they wish to sail. We believe that this would lead to an overloading of the use of pre-conditions. In the present setting of SFOAL we intend the antecedents of modal expressions to perform the descriptive role of trying to enumerate the varying effects an action has in different contexts. In addition, the use of total functions in our transition models means that all actions lead to some scenario - although it may not be known which. Hence, although the expression **I** above does not describe the effect of the sail action in the situation where a ship is already at the port it wants to sail to, this does not mean that performing this action in such a situation should not be defined or is in some way prohibited- simply that its effect is not known. In contrast to this, those formalisms which employ partial functions, VDM and Z for example, in their transition systems would treat the antecedant " $\neg (\text{Atport}(s) = p)$ "

both as defining the sail action and acting as a guard to its performance; in other words, they would conflate the distinction between an action's description and its use to define behaviour.

In order to see this distinction between description and use (or what in future shall be referred to as prescription), consider the following intuitively obvious statements regarding the descriptive and prescriptive aspects of the action of opening a window:

- i. When a window is closed, opening that window results in it being open.
- ii. When a window is open, opening it results in it remaining open.
- iii. If it is sunny it is permitted to open the window.
- iv. If it is raining it is prohibited to open the window.

It should be obvious that the first two of these statements deal with the description of what it means to open a window in the context of the window being either closed or open. In SFOAL we may express i and ii as the following pair of conditional modalities:

- i* $\text{Closed}(W) \rightarrow [\text{Open}(W)] \text{Opened}(W)$
- ii* $\text{Opened}(W) \rightarrow [\text{Open}(W)] \text{Opened}(W)$

Assuming universal quantification over the window variable "W" the meaning of these should be obvious. Turning now to the statements iii and iv, we find that formalisms such as VDM and Z (whose notion of action descriptions is essentially that of Hoare's action and pre/post condition triples [Ho69]) would code these by strengthening the antecedents of *i* and *ii* as follows:

- iii* $(\text{Sunny} \wedge \text{Closed}(W)) \rightarrow [\text{Open}(W)] \text{Opened}(W)$
- $(\text{Sunny} \wedge \text{Opened}(W)) \rightarrow [\text{Open}(W)] \text{Opened}(W)$
- iv* $(\neg \text{Raining} \wedge \text{Sunny} \wedge \text{Closed}(W)) \rightarrow [\text{Open}(W)] \text{Opened}(W)$
- $(\neg \text{Raining} \wedge \text{Sunny} \wedge \text{Opened}(W)) \rightarrow [\text{Open}(W)] \text{Opened}(W)$

We have taken the liberty of using the propositions Sunny and Raining to denote contexts with the obvious interpretation. Note that in order to state the fact that "opening a window is allowed when it is sunny" we have had to strengthen the contextual definition of the action to ensure that it is only defined when it is sunny. Hence the action of opening a window is now only defined for situations in which it is sunny and

the window is either open or closed. This is, we believe, incorrect; the fact that the window is allowed to be opened when it should be quite separate from defining the action of opening the window. Assume for instance that things changed and it became quite unacceptable to open a window when it was sunny, would the action of opening be any different ? Hardly ! But if we adhered to the above strategy of strengthening the precondition/context of the action this is exactly what would happen. This approach becomes even less satisfactory if we are to deal - as we do later - with actions which can affect whether or not other actions are allowed to be performed.

Expression *iv* is also objectionable; in order to prohibit opening the window whilst it is raining the definition of the action is further restricted to be defined in only those situations in which it is not raining. Another, less objectionable approach, would be to define this prohibition as follows:

$$\text{Raining} \rightarrow [\text{Open}(W)] \boxed{\text{FALSE}}$$

Here $\boxed{\text{FALSE}}$ is intended to denote logical falsehood. The idea here is that opening a window in the rain would result in an inconsistent situation and hence have no transition model at all. Although we do not rule out the usefulness of this type of description, it is sometimes too strong; rather than express the reasons for our view here we shall wait until the next chapter.

In contrast to the above, our approach will be to try and separate out the definition of actions and their use in a less confusing and more direct manner. In order to wet the reader's appetite we shall state informally how we propose to model the statments i to iv.

- i $\text{Closed}(W) \rightarrow [\text{Open}(W)] \text{Opened}(W)$
- ii. $\text{Opened}(W) \rightarrow [\text{Open}(W)] \text{Opened}(W)$
- iii $\text{Sunny} \rightarrow \text{Permitted-to}(\text{Open}(W))$
- iv $\text{Raining} \rightarrow \neg \text{Permitted-to}(\text{Open}(W))$

Note how the description of opening the window is not interfered with by information prescribing the behaviour of that action. The formal treatment of modelling this intuitive concept of "permitted to do an action" is the subject of the next chapter wherein we introduce the deontic notions of obligatory and permissible action.

CHAPTER THREE

DEONTIC LOGIC AND BEHAVIOUR

3.1 Introduction.

We ended the last chapter by motivating the need to discriminate between the intrinsic description of the effects actions have and what we earlier termed their prescription, or use. The purpose of this chapter is to describe an extension of SFOAL which accommodates the latter. The extension which we propose is based on the deontic concepts of permission and obligation; however, the resulting system bears little resemblance to classical deontic systems even though it shares many of their aims and objectives. Before we present the details of this extension it will be instructive to explain why we think deontic logic in general, and our specific characterisation of it in particular, is an appropriate tool for the capture of behavioural requirements.

The term *Deontik* was first used by Ernst Mally to name his attempt at creating a logic of *willing* [Wri80]; the basic idea is to develop a logic in which it is possible to state and reason about aspects of the will of a legislator. Here a legislator can be anything from a specific legal institution to a religious order and the notions of ought, prohibition and permission correspond to the legislators *wish* that something be the case, that something not be the case and finally to his *not* wishing that something *not* be the case, respectively. Most of this study originated as an attempt, both by philosophers and logicians, to formally analyse human morality, rights and duty. Unfortunately many of the theorems of Mally's system are counter intuitive; for example "if p ought to be the case then it is the case" is closer to the idea that " p " is necessary since it is not always the case (at least in its natural language use) that if something ought to be the case then it is. Note that here we are using p to denote some state of affairs and not the name of a specific action; we shall return to this point later.

Many other counterintuitive consequences of Mally's axiomatics, a good discussion of which can be found in [FH81], prohibited further logical development of his system. This is not to say that Mally's work was not important; apart from providing a name, many of the counter-intuitive consequences of his axiomatics highlighted old and raised new questions about the form modern day deontic logic was to take.

To some extent the original motivation behind the work of Mally (and others) and the role of creating specifications do have points in common. In the former, the idea of a legislator laying down certain rules of behaviour is not dissimilar to the purpose of a specification, especially if we consider this (in the broad sense) to assert a system of

constraints to which an intended object must adhere. Obviously, the analogy is strongest when we consider the behavioural constraints of such objects. We shall return to this analogy after we have taken a brief look at more recent axiomatics of deontic logic.

Modern deontic logic has its roots in the more general study of alethic modal systems (we have already come across them in the previous chapter) which deal with the concepts of possibility and necessity. But just as other offshoots of modal systems have specialised in dealing with the specific modalities of time (temporal logics), belief, knowledge (epistemic logics) etc. deontic logic has also become a separate area of study in its own right. From here on we shall denote the deontic notions of obligation and permission as O and P respectively, where Op represents the property that p is obligatory and Pp that p is permitted; the alethic concepts of necessity and possibility will be represented (conventionally) as $\Box p$ and $\Diamond p$, respectively.

Current systems of deontic logic owe most, if not all, of their formal underpinnings to the exploitation, in the early 1950s, of the analogy between modal and deontic concepts. Perhaps the best known of these works is the 1951 paper [Wri51] of Von Wright, in which he compares Op to $\Box p$ and Pp to $\Diamond p$. As it turns out many modal theorems, under this interpretation, are also valid for their deontic counterparts, in particular the following:

- i $Op \rightarrow Pp$
- ii $O(p \rightarrow q) \rightarrow (Op \rightarrow Oq)$

Here i is the intuitive requirement that if p is obligatory then it is also permitted; this ensures that nothing which is obliged is also forbidden/not permitted. It is easy to see the intuition in the alethic case since necessity necessarily entails possibility. The second axiom ensures that if the antecedents of obligatory conditionals are also obligatory then so too are their consequents - in other words the principle of detachment or modus ponens is preserved for obligatory properties; we came across the alethic case in the previous chapter. Most systems of deontic logic which incorporate the above axioms are referred to as standard systems of deontic logic.

Although this analogy paid some dividends it was realised early on that there were in fact major differences. One such difference lies in the prescriptive force of obligations; so to say that p is obligatory does not in general entail that p holds currently but rather that p should somehow be brought about. In contrast to this if we say that p necessarily holds we usually intend this to mean that p holds in all possible situations including the

current situation. This alethic property is true of most modal systems and is characterised by the following axiom:

$$\Box p \rightarrow p$$

For the reasons we have just given it is not valid in the corresponding deontic context. Another reason for the divergence between modal and deontic logics has to do with the intuitive scope of the deontic operators. In natural language the notions of permissions and obligations are almost always associated with some verb phrase, the following statements are typical: "Man shall not steal", "Customers are allowed to smoke in this shop" and "All mainframe users must change their passwords at least once a month". The first of these sentences asserts a prohibition, the second a permission and the last an obligation; all three have one thing in common - their aim (in these cases) is to guide the behaviour of people by prescribing which actions they are permitted, prohibited or obliged to perform.

It is rather more difficult to imagine sentences of this kind which do not involve some enactment. To say for example that "it is *permitted* that smoking will occur in this shop" seems a little odd to say the least but is quite acceptable in the alethic context where we may say "it is *possible* that smoking will occur in this shop". It would seem then that the deontic operators are more suited to actions whereas alethic systems are more suitable for formulae which express states of affairs.

There are however problems with this approach; most of these shall not really concern us here but we shall mention them briefly for the sake of completeness. A major problem and one that has troubled philosophers and logicians alike has been the relationship between logical negation and actions. What, for example, is the negation of an action: is it itself an action or is it a prescription not to do an action ? This raises two further questions: if it is the former "what is a suitable formalisation of such actions ?" and if the latter "what does it mean not to do something ?". As far as we can tell no one has adopted the first approach and the second is rife with controversy, involving such notions (and their interrelationships) as action being omitted, refrained from and foreborn to mention but a few; the reader is referred to [Tal85] for further details.

Another issue thought to be problematic at the time involved the restriction of iterating the deontic operators; the expression $O(Pa)$ for example is easily seen to be nonsensical if we observe that Pa is not an action but an assertion. However, it is precisely this kind of iteration between the operators which is sometimes required; think for example of

statements such as " machine-one is permitted to oblige machine-two to perform action 'a' ". As a result of such anomalies both Von Wright [Wri81] and Castenada [Cas72] have advocated discriminating between deontic operators which have actions as their scope and those which deal with formulae expressing states of affairs.

Given these problems, most logicians - even those such as Von Wright who instigated the action based approach - broadened the scope of the deontic operators to include general assertions about states of affairs. Thus, given a sufficiently rich base assertion language, permission to do some action would be construed as applying the permission operator to some sentence incorporating a verb phrase. A typical sentence of this kind would be $P(\text{Jack to drive})$ to be read as "it is permissible for Jack to drive". More general assertions not involving actions could also be made, consider for example the expression $P(\text{Jack is married to Mary})$ read as "it is a permissible state of affairs if Jack is married to Mary". Note that the action of Jack marrying Mary is implicit and the focus is on their static marital status; there remains, however, the implicit assumption that some action is permitted in order to achieve this permissible marital situation. This almost alethic interpretation of deontic logic opened the way for a host of deontic logics - mostly through the result of reinterpreting and modifying various alethic modal systems. The interested reader is referred to either [Che80] or [Wri63] for a more detailed account of such systems.

It is interesting to note that more recently there has been a reversion to actions as the scope of the deontic operators. This is due mainly to two not unrelated reasons: the first is the large (historical) interest that many deontic logicians have in analysing and expressing the prescriptions of legal institutions and the second is the recent introduction of such notions into the software engineering arena. The use in both cases has tended to focus on the prescription of desired behaviour via the notion of permissible and obligatory actions - the latter perhaps more than the former. We shall not deal further with the developments of deontic logic for legal systems, except to point out that they have received a fair amount of attention from the logic programming and more general A.I. computer science sub communities. We refer readers interested in the use of deontic logic for legal analysis in general to [Cas82] and [Wri82] and, more specifically, to [LN] and [Nute] for the deployment of logic programming or artificial intelligence in this endeavour.

As we have seen, deontic logic has had a turbulent yet not uninteresting passage of development, starting from its involvement in the analysis of moral issues right up to its

present day application in the computing science arena. In the remainder of this section we shall overview some of the roles deontic notions have played in computing and motivate the need for a sadly lacking general system of deontic logic tailored to these specific needs.

The role of permission, in both hardware and software, is fairly prominent in computer science and serves in the main to impose certain *a priori* constraints on various parts of a system. Such constraints are often useful whenever it is necessary to limit the use of a system: an example of this would be in a large shared database where different users were restricted in what information they could access and/or modify. Restrictions of this type can be modelled as a collection of permitted or forbidden actions (queries can be regarded as a special erotetic category of actions) where each such collection is dependent on a specific user of the database. The reader is reminded of the fact that this type of database view capability modelling was in fact a major motivational force behind this dissertation. Another example of where this authorisation type of model is employed is in operating systems, where processes are usually only allowed access to certain resources depending, amongst other things, on their priority rating in relation to other processes which may also be competing for the same resources.

Permission has also been used in a more implicit form in various programming languages; one only has to think of certain scope rules which permit access to variables or more autonomous constructs of clusters and packages, found in the programming languages CLU [Lis77] and Ada [Ich79], respectively, for examples. In Ada, for instance, the use of a package is to some extent given by the external packages it is permitted to access and in turn is accessible from.

Specification and design languages have also used the deontic concept of permission; Petri nets and Data Flow Diagrams are good examples of this. Both involve graphical syntax in order to convey permitted behaviour, the former emphasising the sequencing of actions and causality and the latter the permissible routes information is allowed to take within a system. Permission has also been used by Hennessy [Hen82] in order to develop a coarser grain of equivalence testing between processes than the conventional notion of bisimulation. Although the basic intuition of these operators (that some action or transition may or must occur) is the same as ours a direct comparison will not be fruitful, given the differing styles, objectives and use of these notions.

Lastly the idea of a guard in the sense of a pre-condition to an action is also employed to

restrict the use of actions in situations defined by that guard; we have already seen examples of this in the previous chapter. Many specification, design and programming languages employ this idea to express how actions should be used but none seem to make the distinction that we remarked upon earlier between pre-conditions being used to define the effects of an action and being used to prescribe the permitted use of an action. We suspect that the reason for this attitude is due, at least for specification formalisms, to a misplaced sense of circumscription. The general idea is that actions whose preconditions are not satisfied in a particular state are considered to be either undefined or as leading to inconsistency. Whilst it is true that both these strategies fulfill a need, neither are fully satisfactory. In the case where actions are considered undefined for states in which their pre-conditions are not satisfied, the action is often modelled as leading to a special undefined state. This does not seem to allow for loose action descriptions, something which may be an advantage for specifications about an underdetermined application domain. It may, for example, be that a specific situation suitable for performing an action is not yet known; should we then automatically restrict future implementations to "totally" undefined states? The operative word here is of course "totally"; our approach is to leave open what effect an action will have in scenarios in which context/pre conditions are not satisfied as an implementation decision rather than force a special undefined state. Hence our position that actions should be modelled as always leading somewhere (total functions in our models) no matter in which scenario they are performed.

This does however raise the question of how to model the prohibition of actions; one option is to describe such prohibitions as describing the action (in the desired states of prohibition) as leading to an inconsistent state. Although this has its uses we feel that it is too strong for all cases, since inconsistency represents a total collapse of useful information (because it entails all information), whereas some forbidden action may only cause a more local anomaly. We shall return to this point, and that of undefined states, in the next section.

Our own approach to the modelling of permitted/prohibited actions involves the more flexible idea of a normative scenario. Briefly, the idea is to divide all scenarios into two kinds, those that are normative and those which are not; a scenario is normative if the special proposition $\mathbf{n}$ is a consequence of it and non normative if $\neg \mathbf{n}$ is. Normative scenarios are intended to denote deontically acceptable scenarios whereas non normative ones are deontically unacceptable. Given this, actions which are permitted are modelled as leading to normative scenarios and actions which are forbidden to non normative

ones; note that non normative scenarios need not necessarily be undefined or inconsistent. There is one slight complication this and that is that permission and correspondingly non permission for actions are only logically defined (in this way) in normative scenarios; hence the definition of permission is only partial and is relative to normative scenarios. Note that this does not mean that actions necessarily lead to undefined or non normative scenarios from non normative scenarios, just that no universal default is forced. In other words it is left open to a particular specification what the definition of permissible/forbidden actions are in specific situations.

This notion of discriminating between desirable and undesirable scenarios by labelling them is an attempt to report on bad behaviour rather than trying to banish it altogether. Experience in building large scale embedded software systems has shown us that it is naive to expect that all of the system's code will be correct, or that all its users will interact with it in the prescribed way. Yet all our specification formalisms are intolerant of this experience; rather than admit that implementations may only partially fulfil the prescribed behaviour of a specification and reflect this, they tend to demand perfect behaviour. We hope that the reader will not misinterpret this last comment; ideally we would also like to see all implementations abide by the behavioural constraints of a specification. Indeed we do not give up the fact that all correct implementations should admit only normative scenarios. Our aim is to reflect reality a little more faithfully and to provide a means to indicate undesirable behaviour in order to avoid it and to report on it should it occur in an integral and uniform manner. We shall take this point a little further in the body of this chapter.

So far we have talked about the behaviour of a system in terms of actions being either permitted or forbidden; deontic logic offers us an additional notion, that of obligation. The usefulness of this concept in software engineering has been quite forcefully made in [ML85] wherein the authors describe a variety of examples in which the role of this notion is quite essential. As a starting point the purpose of obligations can be motivated by reality; as an illustration think of the act of borrowing something - associated with this act is an intrinsic obligation to perform another act, namely to return the borrowed object. Note that it would be too weak to model this requirement as a permission to return the borrowed object - we need to have something a little stronger if we are to capture the force of *should* rather than *may*. In computing we often feel the need for this more forceful notion; for example, in a distributed database environment it is often necessary to "lock" down certain data before performing an update after which there is an implicit obligation to unlock this data. It would be unwise to model this as simply

being permitted to unlock the data as this would imply that we did not in general mind if the data were permanently locked.

If we were to generalise from these two illustrations it would seem that all we needed was some kind of operator corresponding to "eventually" (the temporal interpretation of possibility). In truth we would need a whole host of operators ranging from those which expressed a mild wish of compliance to those which expressed a more urgent and forceful command of obedience. In addition we would also have to contend with what to do if what we had prescribed *should* happen, did not; and the related task of expressing how urgent the force of the *should* is to be - perhaps by imposing time limits or precedences ? We shall not attempt to deal with all these issues here and will choose instead to deal with only a very strong notion of obligation. To try and do more would, we believe, be premature and ensnare us in the same trap as [ML85] where the authors mention many interesting peripheral concepts, such as punishments for not fulfilling obligations, but have little idea about how to formalise them.

When it comes to characterising obligation the conventional strategy is to express it as the dual of permission, hence just as " $\Box$ " is equivalent to " $\neg \Diamond \neg$ ", " Op " is equivalent to " $\neg P \neg p$ ". In other words some state of affairs p is obligatory, if and only if, it is *not* permitted that *not* p . But as we have seen this is not really applicable in the case where the scope of the deontic operators are actions - mainly due to difficulties in interpreting negative actions. We propose to tackle this problem by the introduction of a new deontic operator defined entirely in terms of permission and negation; we call this operator "Pref". The scope of this operator, just like our permission and obligation operators, are actions where the intended semantics for Pref(a) is that there exists at least one action other than ' a ' which is permitted. Obligation is then characterised by the following two axioms:

$$O(a) \rightarrow P(a) \qquad O(a) \rightarrow \neg \text{Pref}(a).$$

Intuitively these say that if an action ' a ' is obliged then it is also permitted and no other action distinct from ' a ' is permitted. Hence when an action is obliged, it is, necessarily, the only permitted action. Although our present notion of obligatory action is quite strong and has the force of *must do* we do not reject other characterisations of obligation as either uninteresting or useless and propose in the conclusion of this chapter to show how our very strong concept of obligation can be weakened.

However, the reverse, if only one action is permitted then it is obliged, is not valid. The

reason for this lies in our intuition about the dynamic aspects of obligations; we believe that obligations should be modelled as special system interjects which temporarily restrict the system from its ordinarily permissible behaviour and force it to perform the obligatory action in question.

So far we only have a semantic characterisation for this dynamic aspect of obligations and hence our logic is incomplete in this respect. This is hardly suprising when, as we shall soon see, this semantics is expressed in terms of transition constraints at the level of models and incorporates the same difficulties as trying to axiomatise a frame rule/assumption for general transitions. It is, however, worth noting that an alethic style duality between permission and obligation can be recovered if we are prepared to give up this more dynamic perspective. In this case the following equivalence would hold

$$O(a) \leftrightarrow (P(a) \wedge \neg \text{Pref}(a)).$$

There is however a major difference between this and the alethic duality, $\Box \leftrightarrow \neg \Diamond \neg$; in the latter case we are dealing with states of affairs and only implicitly (via the idea of an accessibility relation) with transitions, whereas in our notion of obligation we are dealing explicitly with transitions and only implicitly with their consequences (states of affairs). This distinction is also relevant in choosing between a deontic logic predicated on expressions of states and those predicated on actions. For example, to say " Op ", where p is an expression of state, is to oblige an implicit equivalence class of actions all leading to states in which p holds - there is no finer control over which action is performed. This is a consequence of not talking directly about specific transitions in the logic. On the other hand systems of deontic logic which - like ours - deal with expressions such as Oa , where a is an action, provide a finer grain of control in what exactly is permitted/obliged to happen. This is not to say that the state based kind of operators do not have their uses and we shall see in the next chapter how we can define such operators in terms of our action based ones.

This introduction would not really be complete without mentioning the work of McCarty [McC83]; as far as we know this is the only significant attempt to formalise deontic concepts in the computing arena and is aimed mainly towards the A.I. community. McCarty's aims are different enough from ours for us to simply mention differences in purpose and avoid a detailed comparison; the only common point seems to be that he too adopts the line that permission and obligation are action based concepts. An initial divergence arises in the area of action description; the conventional A.I. approach of using elementary state changes (changes to atomic formulae only) is too restrictive for

conventional specification languages. This is especially true if we consider the related reasons (we gave in the past chapter) for admitting incomplete action descriptions. In his concluding remarks McCarty recognises the simplicity of his language for describing actions but also notes that as this language is made more complex so too will the complexity of his already incomplete axiomatisation.

Further, as admitted by the author, his definition of permission and consequently obligation is intimately related to the definition of the action being permitted or obliged. Hence an action a is permitted only if it produces the desired changes described by that action; in other words rather than try to separate the concerns of describing/prescribing actions McCarty seems to be forcing them together. Perhaps a more secondary point is that McCarty's system does not provide a relationship between state based deontic operators and action based ones. A more detailed critique of McCarty's approach may be found in [Jon87] wherein the author defeats some of McCarty's claims that his system solves some of the paradoxes of deontic logic.

To summarise, deontic logic originated in the more general study of modal logics; however, unlike other offshoots (temporal, belief, action logics etc) the area has not received much attention in the computing science arena. This can partly be attributed to a lack of general awareness by software engineers of what deontic logic has to offer and partly due to the fact that logicians studying deontic systems have not so far looked to the software engineering domain as a potential area of application. It is hoped that the connection will become more established now that there are close links between the legal and A.I. communities. In addition we hope that our work will go some way towards establishing an initial formalisation of deontic concepts suitable for software engineering in general and, in particular, specification theories thereof.

The remainder of this chapter is structured as follows: we shall begin by presenting a formal extension to the logic of the previous chapter; the following section will then describe how this can be used to incorporate the description/prescription distinction argued previously; we shall then illustrate some of the benefits of this extended system with the aid of an example; and, finally, we will conclude by highlighting some shortcomings of this extension and suggest some solutions.

3.2 DAL (Deontic Action Logic)

In this section we shall present an extension to the logic of chapter two; the purpose of this extension, as motivated in the introduction to this chapter, will be to incorporate the deontic notions of permissible and obligatory actions. A specification will then correspond to a theory in this extended logic and the definition of a model for a specification will be extended to reflect this. As before, the extension is given in two parts, a language extension and a set of axioms and rules. We shall only detail the extension here, the reader should augment this with the language (and formation rules), axioms and rules of SFOAL - unless explicitly stated otherwise. We shall refer to this extended logic as DAL (short for Deontic Action Logic).

Language:

Syntactic categories:

- In addition to the action sort **Act** we shall also have the sort **S(Act)**, the elements of which are finite sequences of actions.
- Variables of sort,
 - **S(Act)**, we have the usual infinite collection, and use the symbols $\sigma, \sigma', \sigma'', \dots$ to denote arbitrary non empty finite sequences of actions.
- Predicate symbols,
 - **P** of type **Act**.
 - **O** of type **Act**.
 - **Pref** of type **Act**.
 - **OS** of type **S(Act)**.
- Function symbols,
 - **< >** of type **< Act, S(Act) >**
 - **,** of type **< S(Act), Act, S(Act) >**
- Logical constant **n**.

- Logical connectives, $\llbracket \rrbracket$.

In addition to this we shall assume that the clauses, given in the language of SFOAL, pertaining to quantifiers, variables, and equality apply equally to the new sort $S(\text{Act})$.

Formation Rules:

- Terms:
 - Any constant or variable of the new sort is a term.
- Atoms:
 - The clauses of SFOAL pertaining to the construction of atoms apply to the new predicate symbols, P, O and OS. Equality between terms is also defined on terms of the new sort.
- Formulae:
 - All previous clauses in our description of SFOAL.
 - The logical constant $\mathbf{n}$ is a formula.
 - If σ is a term of sort $S(\text{Act})$ and ϕ is a formula then so is $\llbracket \sigma \rrbracket \phi$.

Here, ϕ is intended to denote any DAL formula.

Axioms:

- (1) $\llbracket \langle \alpha \rangle \rrbracket \phi \leftrightarrow [\alpha] \phi$
- (2) $\llbracket \langle \sigma, \alpha \rangle \rrbracket \phi \leftrightarrow \llbracket \sigma \rrbracket [\alpha] \phi$
- (3) $\mathbf{n} \rightarrow (P(\alpha) \leftrightarrow [\alpha] \mathbf{n})$
- (4) $\neg [\alpha] \mathbf{n} \rightarrow [\alpha] \neg \mathbf{n}$
- (5) $OS(\langle \alpha \rangle) \leftrightarrow O(\alpha)$
- (6) $OS(\langle \sigma, \alpha \rangle) \leftrightarrow (OS(\sigma) \wedge \llbracket \sigma \rrbracket O(\alpha))$

$$(7) \text{ Pref}(\alpha) \leftrightarrow \exists \beta (\neg (\beta = \alpha) \wedge P(\beta))$$

$$O1 \quad O(\alpha) \rightarrow P(\alpha)$$

$$O2 \quad O(\alpha) \rightarrow \neg \text{Pref}(\alpha)$$

Note that we have stuck to our previous convention of using α and β for actions and φ to denote any arbitrary (DAL) formula. The only new variables are $\sigma, \sigma', \sigma'' \dots$ which are used to represent action sequences. The definition of proof remains as before - except that now proofs are carried out relative to the extended language and axioms of DAL. Again, note that FOL proof steps refer to normal first order logic manipulation.

The reader will have noticed that we can replace axioms **O1** and **O2** with the more complex expression, $\forall \alpha \ O(\alpha) \rightarrow (\forall \beta \ P(\beta) \leftrightarrow \beta = \alpha)$; our reasons for not doing this will become clear towards the end of this chapter when we shall comment on how the definition of the Pref operator can be modified in order to describe weaker notions of obligation than we deal with here.

A specificand written in DAL will correspond to a DAL theory, where the definition of a theory is that which was given earlier, the only difference being that the logic is no longer SFOAL but DAL. Note that theories which denote specificands retain the special closure under the static interpretation of our necessitation rule. Next we define what a model is for a specification written in DAL.

DAL Models

The definition of a model structure remains unaltered except that we are now dealing with the logic DAL instead of SFOAL. Hence in what follows **sp** will denote a specification written in DAL, that is a DAL theory. $\mathcal{D}$ is the collection of scenarios associated with **sp** (defined as before except we are dealing with DAL theory extensions) and $\mathcal{F}$ a collection of functions, again as described before, one for each action name in the **Act** category of **sp**. However the definition of a model for a specification **sp** written in DAL is extended, as shown below.

Model Structures:

A *model structure* for a DAL specificand **sp** is given by a pair $\{ \mathcal{D}, \mathcal{F} \}$ where

$\mathcal{D}$: is a set of deontic scenarios, where a scenario is a theory extension of the specificand **sp**. As we shall see later this collection of scenarios is partitioned into two kinds, normative and non normative.

$\mathcal{F}$: is a collection of typed functions, one for each action name in the syntactic category **Act**, of the specification **sp**. The type of such functions is

$$f_{\alpha} : \mathbf{t}(\alpha) \times \mathcal{D} \rightarrow \mathcal{D}$$

where $\mathbf{t}(\alpha)$ denotes the type of the action name α , and $\mathcal{D}$ is the domain of potential scenarios for the specification **sp**.

Models:

A *model structure* $\{ \mathcal{D}, \mathcal{F} \}$ for **sp** is a *model* for **sp** iff the following conditions are met,

- $\forall s \in \mathcal{D} \quad \forall \alpha \in \mathbf{Act}$
 $\text{if } [\alpha] \varphi \in \mathbf{TH}(s) \text{ then } \varphi \in \mathbf{TH}(f_{\alpha}(s))$
- $\forall s \in \mathcal{D} \quad \forall \alpha \in \mathbf{Act}$
 $\text{if } \neg [\alpha] \varphi \in \mathbf{TH}(s) \text{ then } \varphi \notin \mathbf{TH}(f_{\alpha}(s))$
- $\forall s \in \mathcal{D}, \forall \alpha, \beta \in \mathbf{Act}$
 $\text{if } [\alpha] O(\beta) \notin \mathbf{TH}(s) \text{ then } O(\beta) \notin \mathbf{TH}(f_{\alpha}(s))$
- $\forall s \in \mathcal{D}, \forall \alpha, \beta, \delta, \gamma \in \mathbf{Act}, \forall \sigma \in S(\mathbf{Act}),$
 $\text{if } P(\alpha) \in \mathbf{TH}(s) \text{ and } O(\delta) \notin \mathbf{TH}(s) \text{ and } OS(\sigma) \in \mathbf{TH}(f_{\beta}(s))$
 $\text{and } O(\gamma) \notin \mathbf{TH}(f_{\sigma} f_{\beta}(s)) \text{ and } \neg P(\alpha) \notin \mathbf{TH}(f_{\sigma} f_{\beta}(s))$
 $\text{then } P(\alpha) \in \mathbf{TH}(s) \leftrightarrow P(\alpha) \in \mathbf{TH}(f_{\sigma} f_{\beta}(s)).$

- $\forall s \in \mathcal{D}, \forall \alpha, \beta, \delta, \gamma \in \text{Act}, \forall \sigma \in S(\text{Act}),$
 $\text{if } \neg P(\alpha) \in \text{TH}(s) \text{ and } O(\delta) \notin \text{TH}(s) \text{ and } OS(\sigma) \in \text{TH}(f_\beta(s))$
 $\text{and } O(\gamma) \notin \text{TH}(f_\sigma f_\beta(s)) \text{ and } P(\alpha) \notin \text{TH}(f_\sigma f_\beta(s))$
 $\text{then } \neg P(\alpha) \in \text{TH}(s) \leftrightarrow \neg P(\alpha) \in \text{TH}(f_\sigma f_\beta(s)).$

As before we have used $\text{TH}(s)$ to denote the theory (collection of formulae closed under logical consequence) corresponding to scenario (the theory presentation) s and $\in$ and $\notin$ to indicate whether or not a formula belongs to a theory. The use of f_σ is introduced as a shorthand for the sequence of functions corresponding to the action sequence σ . The definition of these functions is as follows

$$f_\sigma =_{\text{def}} f_{a_n} \circ f_{a_{n-1}} \circ \dots \circ f_{a_1}$$

where σ is the sequence $\langle a_1, \dots, a_n \rangle$, f_{a_i} denotes the function corresponding to the action a_i (as defined above) and $\circ$ denotes functional composition.

There are two essential changes between the previous definition of a model and the one above. Firstly all collections of scenarios for a specification are partitioned into two disjoint exhaustive classes called normative and non normative. A normative scenario is one which is behaviourally acceptable whereas a non normative scenario is one which is behaviourally undesirable. Normative scenarios are distinguished from non normative ones by the logical constant $\mathbf{n}$ - normative ones have the property $\mathbf{n}$ whereas non normative ones have the property $\neg \mathbf{n}$. This may be achieved either by using $\mathbf{n}$ in the description of specific actions and hence indirectly prescribing (non) normative scenarios or in a more direct manner, such as relativising the property of being a (non) normative scenario to another scenario based property. We shall see examples of both in the next section.

Secondly, we have the extra restrictions on normative models which are needed to characterise the dynamic nature of obligations, the reader is referred to the last three model conditions above. The reader should note that none of these conditions are forced by our logic, in order to reflect these conditions in our logic we would need to assert rules of the form

$$\frac{\perp \phi \text{ and } \vdash \psi}{\vdash \phi}$$

where $\perp$ represents the notion of something not provable. Rules of this form are not

generally acceptable for decidability reasons - we shall have more to say about this when we look at what we shall term to be the dynamic aspects of obligations. All we wish to do here is state the incompleteness of our logic with respect to these rules. As we shall see later these conditions may be seen as specific instances of the more familiar frame rule (see the previous chapter) and hence, for reasons similar to those we mentioned earlier, must be dealt with at the model level.

Having said what constitutes a model for a specification written in DAL we would like, in the next section, to explore more fully the meaning of DAL concepts and expressions. We shall throughout the ensuing discussion refer to (and indeed elucidate) the definition of DAL models given above.

3.3 DAL Specifications.

In the previous section we presented an extension of SFOAL called DAL with a view to being able not only to describe the pertinent actions in a system, but also to express the behaviour of a system. The foundation stone of our approach is the notion of a permissible action; if an action is permitted then the change described by that action is "allowed" to occur. Since obligations, to a large extent, are characterised in terms of permissions, we shall deal with permissions before obligations. First though a few comments about the language extension are in order.

We shall deal with the deontic operators (P, Pref, O and OS) later; it is the other (non deontic) aspects of the language which we want to talk about here. A new syntactic category, $S(\mathbf{Act})$, is defined, the category (or sort) of non empty finite sequences of actions. Associated with this new syntactic category we have two functions " $< >$ " and " , " which allow us to create and augment (finitely construct) action sequences.

The function " $< >$ " accepts as argument an action and returns the corresponding unitary sequence whereas the function " , " augments an already existing sequence. As an example consider extending the one element sequence $< \alpha >$ by the action β , the resulting sequence may be denoted $< \alpha, \beta >$. It should be noted that (although it is not necessary for our present purposes) other useful functions such as length (of a sequence) could provide a more complete treatment of action sequences. We think that this is a task best left to the design of specific specification languages. It may for example be that an application requires us to take into account specific orderings of action sequences and hence a specific comparison operator may be needed. It would be pointless to second guess, in this way, the particular needs of any specific application domain.

Finally, the new modal connective $\llbracket \rrbracket$ is used to construct formulae of the form " $\llbracket \sigma \rrbracket \varphi$ " where σ is an action sequence and φ is some DAL formula. The purpose of this new connective is more convenience than anything else and its meaning is taken to be equivalent to the following iterated modality, $[\alpha_1][\alpha_2] \dots [\alpha_n] \varphi$, where the sequence σ in " $\llbracket \sigma \rrbracket \varphi$ " is of the form $< \alpha_1, \dots, \alpha_n >$. Axioms one and two above provide a recursive scheme to this effect.

We are now in a position to discuss the deontic operators proper.

Permissions.

During the life of any dynamic system there will be times when that system is allowed to perform certain actions and not others. In DAL an allowed or permitted action is prescribed using the P operator - of type **Act**, where $P(\alpha)$ denotes the property that the action α is permitted and $\neg P(\alpha)$ represents the fact that the action α is not permitted. To understand the meaning or 'force' of this operator we must look at its definition, given by axioms three and four of DAL.

Axiom three equates the property of having permission to do an action with the observation that that action, when performed, results in a normative state. In other words, authorising the enaction of an action amounts to describing that action as one which (when performed) leads to a normative scenario. But this is only so when the permission for the action is given in a normative scenario; this is easily seen if we observe that the expression $P(\alpha) \leftrightarrow [\alpha] \mathbf{n}$, is relativised to the property of being a normative scenario - indicated by the logical constant $\mathbf{n}$. Note that this means that any action which is performed in a normative scenario and leads to some other normative scenario is also considered to be permitted. Forgetting for the moment the extra complication of restricting the definition of permission to normative scenarios there are several reasons for defining this operator in terms of this special action description.

Ideally, permitting or prohibiting an action should correspond to the imperative notions of being allowed to, or forbidden from, performing an action. Our first thoughts on trying to model this correspondence were to model the scenario transforming functions corresponding to actions as partial functions, the domain of defined application being governed by whether or not an action is permitted. Thus any action which is not permitted has no defined consequence whereas the consequences of permitted actions are defined by action descriptions in the usual way. One disadvantage of this is that not only does a forbidden action have an undefined consequence - contrast this with a totally underdetermined consequence - but it also means that the whole system enters an undefined state of affairs. Hence, in addition to the specific consequence of the action being undefined, performing a forbidden action makes everything else undefined too.

Clearly this is counterintuitive; similar arguments also apply to defining forbidden actions as leading to inconsistency - just because a forbidden action occurs it does not mean that a system is plunged automatically into a state of inconsistency. For us inconsistency represents a total system failure; we believe that consistency should be an

a priori system requirement whilst behaviour should be something defined under this assumption.

For these reasons we reject the two extremes provided by the use of partiality and inconsistency and adopt instead a more middle ground. Our approach is to partition the class of all potential scenarios of a system into two kinds: normative and non normative. A scenario is defined to be normative (non normative) if and only if the logical formula $\mathbf{n}$ ($\neg \mathbf{n}$) is a consequence of its corresponding DAL theory presentation. We then define a permissible action as one which leads to a normative scenario and a forbidden action as one which leads to a non normative one. This definition may be represented by the following DAL equivalences:

$$P(\alpha) \leftrightarrow [\alpha] \mathbf{n} \quad \neg P(\alpha) \leftrightarrow [\alpha] \neg \mathbf{n}.$$

The advantage this has over the other two options is that non normative scenarios are considered neither to be undefined nor inconsistent and hence may be reasoned about in the same manner as normative scenarios. An important consequence of this is that we do not rule out the possibility of forbidden actions being performed but rather label such actions as undesirable. Compare this to the case in which forbidden actions are modelled as leading to inconsistent scenarios and hence (effectively) eliminating such actions from the models of a specification altogether. The rationale for this is that an implementation satisfies a specification if and only if those actions which are permitted can be performed and those which are forbidden cannot. In contrast our approach will be to report on forbidden actions rather than banish them altogether; note that we do not rule out this latter option since it is always possible to code it as $[\alpha] (\neg \mathbf{n} \wedge (\forall \beta [\beta] \neg \mathbf{n}))$. Hence, performing the action α has the effect of entering a non normative scenario from which no normative scenario can be reached - indeed only non normative scenarios are reachable. As we shall see in a moment, our looser approach for dealing with forbidden actions allows us to describe fault tolerant systems.

Returning now to axioms three and four of DAL we see that there is one further complication to our definition of permitted/forbidden action - the definition is relative to (necessarily, only valid in) normative scenarios. This seems sensible given the fact that most things - including permissions to perform actions - remain unchanged (c.f. the frame assumption) with respect to performing an action. As an illustration of this consider a scenario in which the actions a and b are permitted but c is not; then performing c will lead to a non normative scenario, but assuming that c does not forbid a

or b as part of its effect and leaving things as they stand would mean that performing either a or b would still lead to a normative scenario. In other words, the behaviour before performing action c is still assumed desirable after performing it; in general this is not something which we can assume since c is a forbidden action which is not 'supposed' to occur. Our argument is that we can only be sure that a permitted action leads to a normative scenario if we are already in a scenario reached by performing a permitted action. Similar arguments hold for the negative case involving forbidden actions; both these restrictions may be represented in DAL as follows:

$$\mathbf{n} \rightarrow (P(\alpha) \leftrightarrow [\alpha] \mathbf{n})$$

$$\mathbf{n} \rightarrow (\neg P(\alpha) \leftrightarrow [\alpha] \neg \mathbf{n})$$

The first of these is axiom three and the second follows from this and the fact that $[\alpha] \mathbf{n} \leftrightarrow [\alpha] \neg \mathbf{n}$. Note that the latter is a consequence of axiom four and the more general axiom $[\alpha] \neg \phi \rightarrow \neg [\alpha] \phi$.

As far as the other logical connectives are concerned the permission operator can be treated like any other monadic predicate. In particular the standard tautological forms such as $P(a) \vee \neg P(a)$ and $\neg(\neg P(a)) \rightarrow P(a)$ are theorems of DAL. As an illustration of the use of permission and the logical constant $\mathbf{n}$ consider the following (partial) DAL specification of a stack; the actions involved are Push(a natural number) and Pop, and the value of the top of the stack is represented by a scenario varying constant (a non rigid designator) called Top. We shall not deal with the description of these actions here but a more complete version of specifying a stack in our framework may be found in either [KMS85] or [Ma86].

$$1 \quad (\text{Top} = 5) \rightarrow \neg P(\text{Push}(10))$$

$$2 \quad \forall n (\text{Top} = 5 \wedge \neg (n = 10)) \rightarrow P(\text{Push}(n))$$

$$3 \quad \forall n (\neg (\text{Top} = 5)) \rightarrow P(\text{Push}(n))$$

In a nutshell this says that the integer ten should never be placed directly above the integer five in the stack. The first axiom would prohibit the operation of placing a ten on top of a stack which had five as its current top element. The second tells us that it is permitted to push any integer other than a ten onto a stack which has a top element of five; finally the third axiom permits the pushing of any integer onto a stack whose top element is not five. Between them the above three axioms fully prescribe when and how the activity of pushing integers onto the stack may or may not be performed. But, say, instead of axiom three we had

$$\forall n (\neg (Top = 5) \wedge \neg (Top = 1)) \rightarrow P(Push(n)).$$

In this case the action of pushing the unit integer on a stack which did not have five as its current top element would be left open. In other words both those models which allow the pushing of '1' and those which prohibit the pushing of '1', on a stack whose top element is five, are acceptable. Hence so are any potential systems (implementations) which respect these models; this is perhaps clearer to see if we note that the following is valid in our system.

$$[\alpha] \mathbf{n} \vee [\alpha] \neg \mathbf{n}$$

Note that although this forces all actions to lead either to a normative scenario or a non normative one we may not know in any particular scenario which it is to be. Again this is a consequence of representing scenarios as theories.

Some deontic logicians, notably von Wright, have used the law as an example of a system which is often noncommittal about the permissibility (or not) of certain actions. In [Wri82] von Wright suggests, for legal systems at least, meta rules to the effect that all actions which are not known to be permissible automatically become forbidden. Our own view on the matter is that if a specification does not determine whether or not an action is permissible then the decision to permit (or not, as the case may be) that action is an implementation one. In other words the specification becomes deontically loose, just as it may be underdetermined for non deontic properties. This is precisely what happens in a legal context. Consider a legal system which is silent about a person's right to perform a certain action, and suppose for the sake of argument that that person carries out this action. If we demand that the legal system in question should comment about the legality of the act then the resulting decision must surely depend on a particular implementation (interpretation) of the system. Things are not as simple as this since legal systems are forever being dynamically reconstituted in the light of new judgements being made and precedents being set. But we hope we have illustrated the point about a specification being silent about the permissibility (or not) of certain actions clearly enough.

Quite apart from its use in characterising permitted actions the logical constant $\mathbf{n}$ can also be used to "statically" bar unwanted scenarios; for example if the expression $\phi \rightarrow \neg \mathbf{n}$ is part of a specification this means that any scenario in which the property ϕ holds is not (and would never be) a normative scenario. Hence, no action (or, as we shall see later, no sequence of actions) could lead to a normative scenario in which ϕ were to hold.

Note that this is less powerful than saying $\neg \phi$ holds in all scenarios as it allows us to delay the question of whether a scenario is normative or not, until we can prove that ϕ holds. Another way to look at this is to observe the fact that as we are dealing with information states it may not be known whether ϕ holds in a scenario s and yet we may wish to consider s to be a normative scenario. As an example of this imagine that we are specifying a medical expert system in which we wish to represent the fact that it is not normative for a creature to be both a reptile as well as a mammal, or

$$\exists \varsigma \text{ Reptile}(\varsigma) \wedge \text{Mammal}(\varsigma) \rightarrow \neg n$$

where " ς " is a variable ranging over the sort "creature". However, as is often the case in the classification of nature, it may not be known (for certain) whether a newly discovered species is mammalian or reptilian. This may be represented by a scenario which contains no information about whether or not a creature is either reptilian or mammalian. Taking ϕ to be

$$\exists \varsigma \text{ Reptile}(\varsigma) \wedge \text{Mammal}(\varsigma)$$

then $\neg \phi$ is

$$\forall \varsigma \neg \text{Reptile}(\varsigma) \vee \neg \text{Mammal}(\varsigma)$$

Comparing $\phi \rightarrow \neg n$ with $\neg \phi$ we can note that in the latter we are stipulating (*a priori*) something about the nature of all scenarios, namely that no creature can be both a reptile and a mammal, whereas in the former we are stating that all scenarios in which a creature is *found* to be both a reptile and a mammal are non normative. The latter precludes any scenario which has creatures that are both reptiles and mammals whilst the former allows this possibility but labels such scenarios as non normative. Again this is a consequence of our general approach - to model erroneous behaviour by reporting it, rather than prohibiting it.

Perhaps the most interesting and powerful feature of our characterisation of the deontic operator 'P' is that it takes the form of an action description. This allows us to describe actions whose purpose is to alter the permission structure of a system in a particularly elegant manner. Remembering that the permissibility of an action is a scenario dependent property, we can describe actions which are indicative of new permission structures in the usual way; for instance if the action α leads to a normative scenario then

$$[\alpha] P(\beta) \text{ is equivalent to } [\alpha][\beta] n$$

and describes an action α , which - when performed - results in a scenario in which it is

permitted to perform the action β . Note that the observations we made earlier about the implicit underdeterminism of actions apply equally to actions which alter permissions. We have already hinted at how explicit underdeterminedness of actions which alter permissions can be catered for when we discussed the issue of scenarios which were undecided (incomplete) as to whether or not permission for a certain action existed. To reinforce this point consider the following action description

$$[\alpha] P(\alpha) \vee P(\beta)$$

which, assuming the current scenario is normative and α is permitted, can be rewritten as

$$[\alpha]([\alpha] \mathbf{n}) \vee ([\beta] \mathbf{n})$$

It should now be easy to see that performing α results in a scenario in which either performing α results in a normative scenario or performing β results in a normative scenario - or both.

Given this ability to describe actions which allow us to change normative structures it is not difficult to see how complex behaviour patterns can be constructed. Later we shall present a concrete example of this, but for now we shall press ahead with our discussion of the other deontic operators.

Refrain.

So far our discussions of whether or not actions are permitted have been of a positive flavor; however, it is sometimes useful to be able to say when the performance of some action can/may be avoided. There are many deontic concepts (all subtle variants of each other) which attempt to capture this idea. Labels such as Omit, Forebare and Refrain have all been used in the literature to try and capture the idea that an action is not performed; a good comparison of these can be found in [Tal85]. The problem which all of these notions try to tackle is the idea of a negative action. As an example of a negative action consider the sentence "Mary is given permission not to cook the meal next Friday"; ignoring the temporal reference what sense can we attribute to the action of **not** cooking the meal ? Is the action of not cooking the meal also an action and if so what is its description ?

The question of negative actions has troubled philosophers for some time now and it is not our aim to provide a general answer to this; we shall instead restrict our attention to

how negative actions can be dealt with in the context of requirement specifications. The "solution" we propose comes about by analysing what it means to be *permitted* (or not) to **not** do some action. We think that being permitted not to do an action, α , is equivalent to saying that there *exists* some action other than α which is permitted. The main assumption behind this position is that in order not to do some action α some other action must be allowed. At first we thought that there was a need for an independent refrain operator, but later discovered that (for our present purposes) it is enough to restrict ourselves to talking about whether or not it is *permitted* to refrain from an action. To this end we introduce the operator 'Pref' defined in axiom seven of DAL.

As an illustration of the use of this operator consider the following expression in which the action Hire-Emp is intended to model the recruitment of personnel:

$$\forall \rho \text{ Pref}(\text{Hire-Emp}(\rho)).$$

The literal meaning of this is that it is always permitted to do something other than hiring a person - whoever the prospective candidate may be. If we are to try and say this using the classical connective $\neg$ we have two options open to us; we could either try

$$\forall \rho \neg P(\text{Hire-Emp}(\rho)) \quad \text{or} \quad \forall \rho P(\neg (\text{Hire-Emp}(\rho))).$$

The second option is problematic because like everyone else [Wri80] [McC83] we do not know what sense to make of negative actions. The first option is also unsuitable given that it forces us to not employ anyone at all. But perhaps we are missing the point, why use " $\neg$ " at all? Is it not sufficient to reflect this company's policy by writing

$$\forall \rho P(\text{Hire-Emp}(\rho)) ?$$

We think not. What this says is that it is permitted to (that is we *may*) employ every person - but does this not inherently entail the property that we may not act on our permission to hire and hence always retain the option of not hiring those people we do not want. Our answer to this line of reasoning is more one of conviction than anything else. In our defence we can only say that what we are suggesting is that the act of refraining should be more positive than passively *not doing*.

The trouble arises due to a confusion between doing nothing, in the passive sense of no action occurring, and of doing the null action, something more positive. We do not consider the idea of doing nothing in the passive sense to be the same as an action happening. On the other hand if we made available a null action, this may then be treated on a par with any other action and is subject to the same deontic prescriptions of being

permitted, refrained from, obliged etc.

If we do not distinguish between these two senses of not doing, it becomes almost impossible to prescribe when a system should **do nothing** (the "null" action) as opposed to **not doing** anything - including not doing the "null" action. Returning now to our "hiring" example, we can observe that permission to hire everyone says nothing about having permission to do anything else (including the "null" action), whereas permission to refrain from hiring anyone stipulates that there is at *least* one alternative (permissible) action. Whether it is the "null" action or some other is immaterial.

As another example consider the following, written as part of a requirement specification for a Network protocol system:

Timeout(Message) $\rightarrow$ Pref(Receive(Message))

This can be read as "When a message is timed-out (considered to be overdue for arrival) then the system is allowed to refrain from receiving it". This should not be too hard to make sense of. What the expression ensures is that the system does not idly waste time waiting to receive timed-out messages but has the option of taking other more positive action, for instance sending a non acknowledgement message. Note that this does not mean that when time-out occurs it loses the permission to receive the message. It is consistent to have either Pref(α) and P(α) or Pref(α) and $\neg$ P(α) holding in a scenario at the same time.

It should not now be difficult to see that refraining from an action provides a different kind of negation on the permissibility of actions than the classical $\neg$ connective. The notion of refraining from an action will also play a part in how we characterise the idea of action being obliged (or not) to happen. It is to such a characterisation to which we now turn.

Obligations.

There are two aspects to our characterisation of what it means to be obliged to do an action: the static aspects which are captured by the two axioms **O1** and **O2** and the dynamic aspects which are modelled by certain conditions on DAL models. We will look at the static aspects first.

Static aspects.

As we observed earlier, as a consequence of the history of deontic logic, the notion of an obligation is usually defined as the dual of permission - in much the same way that necessity and possibility are defined as duals in most alethic logics. Although this seems to work well with alethic logics (those which deal with the notions of possibility and necessity) we shall see that this approach is not appropriate for the purposes of dealing with the idea of obligatory actions.

For us, an obligatory action is one which must be performed, where the sense of 'must' is not that which has the force of 'will' happen but rather that if some action does happen, it can only be the obliged one. In terms of the behaviour of a system this means that whenever the system is in a scenario in which an obligation to do an action α exists then the very next scenario the system will enter will be the one which is arrived at by performing the action α . Having stated the intended meaning of obligatory actions it is time now to see how we can capture this in our logic. As a first attempt we could try the duality principle mentioned earlier; in alethic systems this takes the form $O \phi \leftrightarrow \neg P \neg \phi$ where ϕ is some formula indicative of some state of affairs and O and P are the obligation/necessity and permission/possibility operators, respectively. The closest we can come to this characterisation, given the fact that we are concerned with actions rather than states of affairs - and the discussion of negative actions above - is the following:

$$O(\alpha) \leftrightarrow \neg \text{Pref}(\alpha)$$

which reads as "an action α is obliged if and only if there is no other action which is permitted". However, this only captures part of our intuition; although it bars the occurrence of all other actions it says nothing about whether or not the obliged action is allowed. Hence it is quite consistent to envisage scenarios in which a system is obliged to do an action but does not have permission to do so; clearly this is not what we intended. A simple solution to this anomalous situation is to make sure that whenever an obligation for an action exists, so does permission for that action. One way to ensure this is to incorporate it into our logic as an axiom, that is add the following as an axiom:

$$O(\alpha) \rightarrow P(\alpha)$$

But if we consider this together with the equivalence above we get the unfortunate situation that whenever *no* action, other than α , is permitted then the action α is permitted. Clearly this is intuitively unacceptable as it commits (*a priori*) a system to have at least one permissible action. Although this seems reasonable in the context of

global and anonymous system actions it will become less acceptable after the introduction of agents. Briefly, there may be systems where certain agents in specific situations are prohibited from performing any action and must remain so disabled until other agents grant them certain permissions (by themselves performing actions). A good example of this can be found in multi-tasking computer operating systems, such as UNIX (Unix is a trade mark of Bell Labs USA), where users (agents) are from time to time disabled. The problem, of course, lies in the equivalence

$$O(\alpha) \leftrightarrow \neg \text{Pref}(\alpha)$$

which is too strong for our purposes, the entailment from left to right is, of course, correct - the reverse entailment, however, is not right. Reading from right to left in the above we see that not having a permissible alternative to some action commits us to being obliged to do that action. Removing the equivalence and replacing it with the axiom

$$O(\alpha) \rightarrow \neg \text{Pref}(\alpha)$$

seems more appropriate; this together with the axiom $O(\alpha) \rightarrow P(\alpha)$ captures exactly the intuitive description of obliged actions that we gave earlier.

To summarise then, an obligation to do an action entails the fact that that action is permitted and further that all other actions are not permitted. Given this, we might also consider the possibility of asserting an axiom to the effect that whenever we have permission to do a certain action and no permission to refrain then we are obliged to do that action. After we describe the dynamic aspects of obligations we shall see why this is not appropriate.

Dynamic aspects.

Natural language provides a very powerful hint as to the dynamic aspects of obligations - especially in their relationships to permissions. In normal usage it is common to say that a permission is granted or revoked whereas obligations are said to be either incurred or discharged. The force of this usage is deeper than mere syntactic form; it belies a strong intuition about the inherent persistence of permissions and, in contrast, the inherently transitory nature of obligations. Hence, when a permission is granted, the default is to assume it persists until it is revoked or derogated. The important point to note here is that it is commonly accepted that the default is to assume that the permission is still operative unless we have knowledge to the contrary. So when we say that "Sam

is permitted to go swimming", we assume that Sam will assume that he may go swimming until we take this privilege away from him. In comparison assume that Sam incurs an obligation to go swimming - as the result of an earlier promise say - in this case our intuitive default tells us that once Sam has been swimming he will have discharged his obligation and so neednot feel obliged to go again. We stress the importance of the word default. It may for example be that Sam incurs a similar obligation as a result of the first - he makes the same promise again whilst he is swimming. It is as though an obliged action carries with it its own derogation.

If obligations are not considered to be transitory but are attributed the same persistence as other information, incurring an obligation to perform an action would force us to keep performing that action forever - unless the consequence of performing the obliged action included revoking this obligation. It is, we think, more appropriate to think of obligations as intrinsically transitory, their presence lasting only whilst the obliged act has not been performed.

Apart from trying to capture this transitory aspect of obligations there is also another (related) difficulty we must overcome. Assume that there exists a situation in which a number of actions are permitted and the result of performing one of them leads to a situation in which an obligation is incurred. Now let us suppose that the action which results in the obligation is actually performed, then according to our two (static) obligation axioms all permissions pertaining to actions other than the obliged action are revoked. Now, if in this new situation, the obligatory action is performed, the current information about which actions are permitted (and which not) will be carried forward (as a consequence of the frame rule) into the resulting scenario - unless specifically altered by the performance of the obligatory action. But since only one action is permitted in any situation in which an obligation exists, this means that most situations resulting from performing obliged actions will be ones in which no permission for any action exists other than the action which was obliged. Clearly this is not desirable.

It may be easier to visualise this in diagrammatic form; however, before we do so it will also be useful to introduce some new terminology. From now on we shall refer to the permissibility of actions relative to a particular system scenario as a p-structure. Thus a p-structure is a scenario specific concept which defines which actions are permissible or not and hence (implicitly) about those actions about which we neither know whether they are permitted or are not. Returning to the points in hand consider a system which has four extralogical actions, a, b, c and d, and the following system scenario properties:

$P(a), \neg P(b), P(c), \neg P(d),$

$[a] O(b), [a][b] \neg P(d), [c] \neg P(a)$

For convenience we shall label this scenario s . Assume now that the action a is performed in s , resulting in the scenario $a(s)$; then this resulting scenario would have the following properties

$\neg P(a), P(b), \neg P(c), \neg P(d), O(b),$

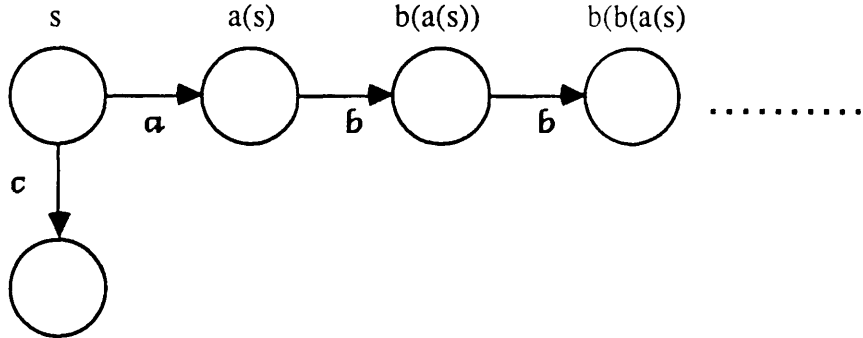
$[a] O(b), [a][b] \neg P(d), [c] \neg P(a), [b] \neg P(d)$

But the only action which can be performed in this scenario is b and performing it results in a scenario, $b(a(s))$, with the following properties

$\neg P(a), P(b), \neg P(c), \neg P(d), O(b),$

$[a] O(b), [a][b] \neg P(d), [c] \neg P(a), [b] \neg P(d)$

again in this scenario the only action we can perform is b . We may represent these scenario changes diagrammatically as



where the scenarios are denoted as nodes and permitted actions label the arcs. It seems then that we have reached a deadlock situation. We have already seen the reasons for this; the fault lies with treating obligations on the same footing as permissions, or any other information. The three extra constraints on models (that we gave above) are supposed to prevent this kind of undesirable behaviour.

The first constraint limits the occurrence of obligations to those scenarios in which they are *explicitly* prescribed by a particular specification. Its literal reading is: "If in a scenario s it is not the case that, for a particular action α , an action description

prescribing the fact that another action β will occur after α is performed, then actually performing α in the scenario s cannot result in a scenario in which the action β is obliged to happen". In symbols this is:

- $\forall s \in \mathcal{D}, \forall \alpha, \beta \in \text{Act}$
 $\text{if } [\alpha] O(\beta) \notin \text{TH}(s) \text{ then } O(\beta) \notin \text{TH}(f_\alpha(s))$

Thus in the above illustration because we do not have the action description $[b] O(b)$ in the scenario $a(s)$, the obligation to do b in the scenario $b(a(s))$ also cannot exist. Although this goes part way to capturing the dynamic nature of obligations, it does not go far enough. We still have the problem of trying to recover the permission structure that existed before the obligation was incurred. It is the task of the last two model constraints to capture this kind of behaviour. It is to an analysis of these two constraints to which we now turn.

We start our analysis by referring to our previous illustration; in our initial scenario s we had the following p-structure

$$P(a), \neg P(b), P(c), \neg P(d)$$

After performing the action 'a', we arrived at the following p-structure:

$$\neg P(a), P(b), \neg P(c), \neg P(d)$$

entailed by incurring the obligation to do b ; performing b then preserves this state of affairs. Intuitively what we would like is to have the p-structure that existed before incurring the obligation reinstated in the scenario reached by performing the obliged action. Given this, it would seem reasonable to impose a pair of constraints along the lines of

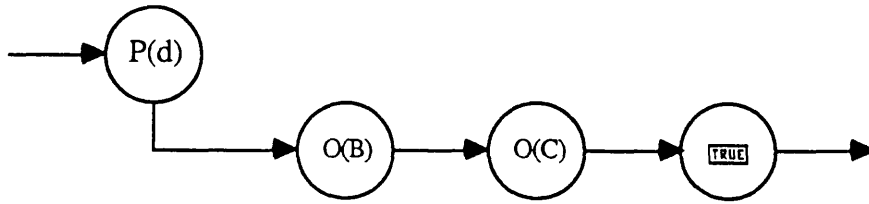
$$\begin{aligned} &\forall s \in \mathcal{D}, \forall \alpha, \beta, \delta \in \text{Act} \\ &\text{if } P(\delta) \in \text{TH}(s) \text{ and } O(\alpha) \in \text{TH}(f_\beta(s)) \text{ then } P(\alpha) \in \text{TH}(f_\alpha f_\beta(s)). \\ &\text{if } \neg P(\delta) \in \text{TH}(s) \text{ and } O(\alpha) \in \text{TH}(f_\beta(s)) \text{ then } \neg P(\alpha) \in \text{TH}(f_\alpha f_\beta(s)). \end{aligned}$$

The first conditional says: "If a permission to perform an action δ exists in a scenario s , then if a scenario is reached, by performing an action β , in which an obligation to do α exists, then upon performing α the permission to do δ is recovered". The same analysis applies with respect to not having permission to do a certain action - this is captured by the second conditional.

There are two major arguments which the above constraints do not allow for. Firstly, an obliged action may actually have the effect of changing the p-structure of the scenario it is initiated in and, secondly, whenever a sequence of obligations (one followed immediately by another) occurs, the rules do not carry over the p-structure of the scenario in which the sequence started. To see what we mean more clearly consider the two element obligation sequence below

$$P(d) \wedge [a] (O(b) \wedge [b] (O(c) \wedge [c] \boxed{TRUE}))$$

which can be seen diagrammatically as:



Here we see that according to the proposed constraints above, after performing the action c the resulting scenario would contain only the permission to perform b and no permission to perform any other action. Whereas what we would have liked is for the permission to perform d to be preserved.

It is also worth noting that it is meaningless to prescribe a change in a p-structure whilst at the same time trying to impose an obligation. Take for example the following DAL expression

$$[Sail(Voyager, Dover)] (O(Unload(Voyager, Tea)) \wedge P(Sail(Voyager, Boston)))$$

which says that after sailing to Dover the Voyager (the name of a ship) is obliged to unload any tea she may be carrying and is also permitted to sail to Boston. Clearly we have an inconsistency here since the static aspect of what we mean by an obligation entails the fact that the Voyager is not permitted to sail to Boston (immediately) after sailing to Dover. What is probably intended by the writer of such an expression is that after unloading tea at Dover the Voyager is permitted to sail for Boston. This would be written as

$$[Sail(Voyager, Dover)] (O(Unload(Voyager, Tea)) \wedge [Unload(Voyager, Tea)] P(Sail(Voyager, Boston))).$$

Another place in which the distinction between changing p-structures and imposing

obligations appears is when we try to derogate a permission at the same time as imposing an obligation. If we consider the following expression:

$[Sail(Voyager, Dover)] (O(Sail(Voyager, Boston)) \wedge \neg P(Unload(Voyager, Tea)))$

we see that asserting that the Voyager is not permitted to unload tea at Dover is an implicit consequence of the fact that she is obliged to sail to Boston. Hence in place of the inconsistency which we found before we encounter redundancy.

Both these examples serve to illustrate that it is important to make sure that we don't confuse the powerful (but transitory) nature of obligations with the more local (and inherently persistent) effects of changing specific permissions.

As we have seen the simple-minded constraints we proposed above failed for two reasons; they did not take into account the fact that when obliged actions are performed they could themselves change p-structures and they did not cater for sequences of obligations. The following model constraints cater for both previous failings:

- $\forall s \in \mathcal{D}, \forall \alpha, \beta, \delta, \gamma \in Act, \forall \sigma \in S(Act),$
 $\underline{\text{if}} \ P(\alpha) \in TH(s) \text{ and } O(\delta) \notin TH(s) \text{ and } OS(\sigma) \in TH(f_\beta(s))$
 $\text{and } O(\gamma) \notin TH(f_\sigma f_\beta(s)) \text{ and } \neg P(\alpha) \notin TH(f_\sigma f_\beta(s))$
 $\underline{\text{then}} \ P(\alpha) \in TH(s) \leftrightarrow P(\alpha) \in TH(f_\sigma f_\beta(s)).$
- $\forall s \in \mathcal{D}, \forall \alpha, \beta, \delta, \gamma \in Act, \forall \sigma \in S(Act),$
 $\underline{\text{if}} \ \neg P(\alpha) \in TH(s) \text{ and } O(\delta) \notin TH(s) \text{ and } OS(\sigma) \in TH(f_\beta(s))$
 $\text{and } O(\gamma) \notin TH(f_\sigma f_\beta(s)) \text{ and } P(\alpha) \notin TH(f_\sigma f_\beta(s))$
 $\underline{\text{then}} \ \neg P(\alpha) \in TH(s) \leftrightarrow \neg P(\alpha) \in TH(f_\sigma f_\beta(s)).$

Both these constraints govern the way in which p-structures are recovered after the discharging of a (sequence of) action(s). Since the only difference between them is that the first deals with permitted actions and the second deals with those not permitted we shall only discuss the first, taking it for granted that the reader can supply a similar explanation for the second.

The constraint is in the form of a conditional, the antecedent of which consists of five premises; before we discuss these premises we shall look at the consequent. This says that the action α is permitted in the scenario s if and only if it is permitted in the scenario

reached by discharging the obligation sequence σ , incurred by performing the action β . The reason for an equivalence (instead of a simple entailment) is that not only do we want to preserve permissions and non permissions, but we also wish to preserve the state of incompleteness as regards permissions. This means that scenarios reached by performing (sequences of) obligations cannot arbitrarily decide on the permissibility (or not) of an action which previous to the obligation (sequence) was not known (not provable either as permitted or not permitted).

It is time now to view the constraints that we have placed on this consequent and we shall deal with the premises of our antecedent one at a time.

$$P(\alpha) \in \mathbf{TH}(s)$$

This is the simplest of the five to explain. All that we need to say is that in order to preserve the permission to do an action (after discharging an obligation, or sequence thereof) we must have permission to do the action in the first place.

$$O(\delta) \notin \mathbf{TH}(s)$$

The purpose of this premise is to avoid preserving the p-structures pertaining at the times when obligations exist. As we saw above, these are only transitory p-structures and it is to avoid the propagation of these imposed p-structures that we are formulating these constraints in the first place. The premise restricts the application of the model constraint to those scenarios in which no obligation exists.

$$OS(\sigma) \in \mathbf{TH}(f_\beta(s))$$

This premise kicks us off into an obligation sequence; what it names is an obligation sequence σ starting in the scenario reached by performing some action β in our initial scenario s .

$$O(\gamma) \notin \mathbf{TH}(f_\sigma f_\beta(s))$$

We need this premise in order to select only the maximal obligation sequences originating in $f_\beta(s)$. If we selected any arbitrary size obligation sequence then the rule would be applicable to scenarios whose p-structures represented the imposition of obligations. If this were the case, then it is possible that the (non-transitory) p-structure we are interested in recovering clashes with the presence of an obligation, resulting in inconsistency in practically all cases.

$$\neg P(\alpha) \notin \mathbf{TH}(f_\sigma f_\beta(s))$$

It is the job of this premise to ensure that permitted actions, which as a consequence of performing the obliged sequence have been changed (directly or indirectly) to not being permitted, are not "overwritten" when the p-structure (existing before the obligation sequence was incurred) is reinstated.

The conjunctive effect of all these premises in the antecedent of our constraint deals effectively with the two major problems we highlighted earlier. Premises three and four deal with problem of having sequences of obligations, whereas the fifth (last) premise takes into account the fact that obliged actions (or more correctly maximal sequences of them) can change non-transitory p-structures.

This concludes our description of the above constraints and hence our view of the dynamic aspect of obligations. Given this view it should now be easy to see our reasons for not accepting the suggestion that

$$i \quad (\neg \text{Pref}(\alpha) \wedge P(\alpha)) \rightarrow O(\alpha)$$

should be valid. If it were valid the model constraints governing how deontic information is preserved when incurring an obligation would also apply to scenarios in which we had a persistent p-structure of the form: $\neg \text{Pref}(\alpha) \wedge P(\alpha)$. Hence to accept the validity of *i* would conflate the distinction between persistent p-structures in which only one action is permitted and transitory p-structures in which we also (by virtue of the static aspects of obligations) have only one permitted action.

In the next section we shall see an example of the use of DAL.

3.4 An Example.

In this section we shall use the deontic concepts developed so far to present and discuss an example specification, showing how DAL may be used to capture the behavioural requirements of a system. We have chosen, for reasons of continuity and comparison, to extend our previous example of a South American shipping company. This extension will take the form of augmenting our previous specification with axioms expressed in DAL. For convenience we shall replicate the original specification (the reader being referred to the previous chapter for an intuitive description of its meaning). Following this, we present the new axioms, describing what they represent and how they alter the original specification. Finally we discuss some deficiencies of DAL as it stands so far and propose some solutions which we look at in the following two chapters of this work.

Here then is the original specification:

Language

Sorts: Ships, Ports, Cargo, Nat, Act.

Constants for:

Ships: Voyager, Discovery.

Ports: Boston, Dover, Bombay.

Cargo: Tea, Steel, Timber.

Variables for:

Ships: s, s', s'',

Ports: p, p', p'',

Cargo: c, c', c'',

Function symbols:

Weight of type < Ships, Cargo, Nat >

Atport of type < Ships, Port >

Total of type < Ships, Nat >

Predicate symbols:

Carries of type < Ships, Cargo >

=_s of type < Ships, Ships >

=_p of type < Ports, Ports >

=_c of type < Cargo, Cargo >

Action names:

Sail of type < Ships, Port, Act >

Unload of type < Ships, Cargo, Act >

Axioms

- (1) $\forall s \text{ Weight}(s, \text{Timber}) \leq 50$
- (2) $\forall s \forall c \text{ Weight}(s, c) = 0 \rightarrow \neg \text{Carries}(s, c)$
- (3) $\forall s \forall c \text{ Weight}(s, c) > 0 \rightarrow \text{Carries}(s, c)$
- (4) $\forall s \forall c \text{ Total}(s) = \text{Weight}(s, \text{Tea}) + \text{Weight}(s, \text{Steel}) + \text{Weight}(s, \text{Timber})$
- (5) $\forall s \text{ Total}(s) \leq 200$
- (6) $\forall s \forall p [\text{Sail}(s, p)] \text{ Atport}(s) = p$
- (7) $\forall s \forall c [\text{Unload}(s, c)] \neg \text{Carries}(s, c)$
- (8) $\forall s \text{ Atport}(s) = \text{Bombay} \rightarrow [\text{Sail}(s, \text{Boston})] \text{ Weight}(s, \text{Tea}) \geq 50$
- (9) $\text{Atport}(\text{Voyager}) = \text{Boston}$
 $\rightarrow [\text{Sail}(\text{Voyager}, \text{Dover})] \text{ Weight}(\text{Voyager}, \text{Steel}) \leq 100$
- (10) $\forall s [\text{Sail}(s, \text{Dover})]$
 $[\text{Unload}(s, \text{Steel})][\text{Sail}(s, \text{Boston})] \text{ Weight}(s, \text{Timber}) = 50$

The new axioms are as follows; note that we have not extended the extra-logical language of the original specification in any way. The only aspect to change has been the underlying logic and in consequence our means of logical expression.

- (11) $\forall s \forall p \text{ Atport}(s, p) \rightarrow \neg P(\text{Sail}(s, p))$
- (12) $\forall s [\text{Sail}(s, \text{Bombay})] P(\text{Sail}(s, \text{Boston})) \vee P(\text{Sail}(s, \text{Dover}))$

(13) $\text{Atport}(\text{Voyager}, \text{Boston}) \rightarrow \text{P}(\text{Sail}(\text{Voyager}, \text{Dover}))$

(14) $[\text{Sail}(\text{Discovery}, \text{Bombay})]$

$[\text{Sail}(\text{Discovery}, \text{Dover})] \text{P}(\text{Sail}(\text{Discovery}, \text{Boston}))$

(15) $\forall s \text{ Atport}(s, \text{Boston}) \rightarrow$

$[\text{Sail}(s, \text{Dover})] (\text{O}(\text{Unload}(s, \text{Timber}))$

$\wedge [\text{Unload}(s, \text{Timber})] \text{P}(\text{Sail}(s, \text{Boston}))$.

It will be appropriate at this point to explain the additional axioms (11) through to (15). The first axiom says that no ship may sail to the port it is already docked at. This seems intuitively sound since it would be bad business practice (ignoring emergencies such as engine failure, with which we do not deal in this simple specification) to return to the same trading point without visiting somewhere else first. It is for this reason that we do not permit ships to sail to ports at which they are already docked.

Independently of good business practice, it seems reasonable that the shipping company would want to impose such a condition given that it is rather difficult to make sense of being able to do something which has no visible effect. Here "visible effect" is measured by the action description of the action in question. In the case of the action of sailing from some port to itself, the action descriptions of the above specification do not alter our information about the shipping company before the action of sailing was performed.

The implication of this axiom at the model level should also be clear - models are restricted to those in which the functions corresponding to the actions of ships sailing from the same port as their destination lead to non normative scenarios.

The next axiom, axiom (12), asserts the property that all ships which sail to Bombay have, upon arrival at that port, permission to sail to either of the other two ports. This is in fact an example of an action whose consequence (of being performed) may dictate a change in the permission structure of the system under consideration. The use of the inclusive "or" connective in the consequent of this action description allows for a number of cases. After any ship 's' performs the action of sailing to Bombay it has either permission to sail to Dover or to Boston or permission to sail to both. The action description is in fact (what we have previously termed) explicitly underdetermined.

The important observation here is that models and implementations in which ships sail to

Bombay, and then have no permission to sail on to Dover or Boston (as there next action) are ruled out and do not satisfy the specification. Whilst on the topic of models and implementations, it might also be worth pointing out that the actual combination of permissions given after sailing to Bombay may be scenario dependent. It may, for example, be the case that if the ship *Discovery* sails to Bombay whilst carrying a cargo of tea, it acquires permission to sail to (the party at) Boston. If on the other hand it is not carrying any tea, it may only be permitted to sail on to Dover. Both these scenario dependent choices fall within the general description of axiom (12). It should not now be difficult to see that the underdeterminism of the action description in axiom (12) behaves like a definite constraint similar to those employed in logic database descriptions [Rei78].

Axiom (13) is less general than the previous two and deals specifically with the ship called *Voyager* and the ports of Boston and Dover. It expresses the fact that whenever the ship *Voyager* is docked at Boston it has permission to sail to Dover. The effect of this at the model level is to be taken in the same manner as any other relativised action description, and this should be clear from previous examples.

The next axiom is similar to axiom (12), in that it involves an action description which changes permission structures, but differs in that it is also an example of what we earlier called an iterated modality. Again, as in axiom (13), it deals with specific actions, ports and ships. In a sentence, the axiom says that "If the ship *Discovery* sails first to Bombay and then immediately on to Dover (no other intermediate action occurs), then it will be permitted (as a consequence) to sail to Boston". Once again, there is nothing special about this, especially if we consider the following equivalent expression:

[Sail(*Discovery*, Bombay)] [Sail(*Discovery*, Dover)] [Sail(*Discovery*, Boston)] $\mathbf{n}$

which reduces axiom (14) to a normal (iterated) action description. Again, the model level interpretation should be self evident.

So far we have dealt only with axioms which describe when action are allowed to be performed (or not, as the case may be). As we have seen these are not too different to ordinary SFOAL expressions; as a consequence of the dynamic aspects of obligations, the next and indeed last axiom is slightly more complicated.

Literally axiom (15) says the following: " If any ship currently docked at Boston sails to Dover, then it incurs (as a consequence) an obligation to unload all its timber and, after

doing so, is then permitted to sail back to Boston". The first thing to note is that the action description involved in this axiom is relativised to those ships that are docked at Boston, that is the context condition is "Atport(s, Boston)". Taking this as understood (we assume that all further discussion presupposes that the ship in question is currently docked at Boston) and turning to the modality itself we see that it can be rewritten as

$$\begin{aligned} & ([\text{Sail}(s, \text{Dover})] \text{O}(\text{Unload}(s, \text{Timber})) \\ \wedge & ([\text{Sail}(s, \text{Dover})] [\text{Unload}(s, \text{Timber})] \text{P}(\text{Sail}(s, \text{Boston}))) \end{aligned}$$

using the fact that actions distribute over conjunctions (c.f. the axiomatics of SFOAL).

The first half of this conjunction says that any ship (reading in the context of the original expression where the variable s is bound to a universal quantifier) sailing to Dover must (is obliged to) unload its cargo of timber. This, due to the axioms O1 and O2 of DAL, entails

$$\begin{aligned} & [\text{Sail}(s, \text{Dover})] \text{P}(\text{Unload}(s, \text{Timber})) \\ \wedge & [\text{Sail}(s, \text{Dover})] \neg \text{Pref}(\text{Unload}(s, \text{Timber})) \end{aligned}$$

which means that the scenario reached by the ship s performing the action of sailing to Dover is one in which the only permitted action is for the ship s to unload its timber.

Looking now at the second half of the original conjunction we see that if a ship unloads timber after sailing to Dover then (as a consequence) it also acquires permission to sail to Boston. It is now easy to make sense of the modality as a whole; after sailing to Dover a ship is forced to unload its timber which (apart from the effect of making sure that in the next scenario the weight of timber is zero; see the original SFOAL specification, axiom (7)) when performed permits that ship to sail to Boston.

In the above analysis of axiom (15) we have only considered the static aspects of incurring obligations. Looking now at their dynamic nature, we see that after a ship has sailed to Dover, all actions other than that ship unloading its timber are not permitted. This means that all other ships have no permissible actions; clearly this is undesirable since all the behavioural information about the other ships is lost as a consequence of incurring an obligation. The dynamic constraints governing obligations ensure that the p-structure that existed before the occurrence of the action which incurred the obligation is reinstated. This is done in such a way that any change in p-structure caused by the obliged act is taken into account. In this case this means that after unloading timber the

ship in question retains permission to sail to Boston irrespective of whether it had permission to do so before sailing to Dover.

Having explained each axiom in isolation we are now in a position to talk about some more general points.

The dynamic aspects of permissions and obligations.

Earlier we made a claim to the effect that permissions were different in nature to obligations, pointing out that permissions were somehow more persistent than obligations. The time has come to explore this claim more fully and we shall do so in the context of the above example. Our argument will hinge on the presence of the frame axiom described in the previous chapter. The heart of our claim lies in the fact that we think permissions (as well as their negatives) are to be taken as preserved over the occurrence of actions unless the action descriptions force it to be otherwise. Things are not as simple as this and the reader is referred to our discussion of this matter in the previous chapter, where the points made about non deontic information are also relevant for the information regarding permissions. On the other hand, obligations are not supposed to have this default persistency, instead they are temporary interjects that force a system to suspend its current behaviour, execute the obliged action and then resume previous behaviour taking into account the effect of the obliged action. Remember this action may prescribe new behaviour.

Looking at our example we see that there are a number of ways in which we can cause certain permissions to be granted and others to be revoked. For example, the iterated modality of axiom (14) says that if the ship *Discovery* sails first to Bombay and then straight on (without unloading) to Dover it will be given permission to sail to Boston. For the sake of argument, let us assume that the shipping operation in question has been implemented and we are in a scenario *s1*. Further let us name the scenario reached by *Discovery* sailing first to Bombay and then to Dover as *s2*. It follows from what we have just said about axiom (14) that in the scenario *s2* *Discovery* has permission to sail to Boston. Now suppose that the *Discovery* decides to unload some steel and call the resulting scenario *s3*. Will the *Discovery* still have permission in *s3* to sail to Boston ?

The short answer is yes. The rationale, which is slightly longer, is that the action of unloading steel does not in anyway change the p-structure of the scenario *s2*, its only effect (as given in axiom (7) of our specification) is to ensure that the *Discovery* is not

carrying any steel as a consequence of unloading it. It seems to us intuitively right that permissions persist in this way as long as actions do not affect them. If it is the case that some permission is not to persist over a particular action then that action's description should be such that the effect of performing that action revokes that permission. As an example consider what happens if in scenario s2 the Discovery decides to make use of its permission to sail to Boston. If it does sail to Boston it immediately loses its (previous) permission to sail to Boston - since no ship can sail to the port it is currently docked at (noting that the consequence of sailing to a port is that you are also docked there, axiom (6)); see axiom (11) of our specification.

So much for the persistency of permissions; but what about obligations ? Obligations have the reverse dynamic nature to permissions; whereas permissions persist until some action decrees otherwise, obligations disappear after they have been discharged (achieved by performing the obliged action) unless the obliged action has the effect of reintroducing the same obligation. This latter property is a specific form of a more general feature, namely obligation sequences; an obligation sequence occurs whenever an obliged action introduces another obligatory act.

For the case of a unitary obligation sequence (we deal with other cases later) we can take as example a slightly modified version of axiom (15) above. Instead of augmenting the description of unloading timber we just leave the description of the action "Unloading" as it is. The resulting conditional is

$$\forall s \text{ Atport}(s, \text{Boston}) \rightarrow [\text{Sail}(s, \text{Dover})] \text{O}(\text{Unload}(s, \text{Timber}))$$

In addition to this assume that the ship Voyager is docked at Boston in a scenario labelled s1. Now, suppose that it sails to Dover where, according to our new rule, it incurs an obligation to unload its timber. When this obligation is discharged it also disappears thanks to the model constraints which enforce this dynamic aspect of obligations. But, had we attributed the same persistency property as permissions, the obligation to unload timber would persist forever.

The obvious question now should be, why can we not explicitly (wherever necessary) revoke obligations the same we do permissions ? There are two major reasons for this.

Firstly we think that it is not part of the spirit of both the intuitive meaning or the historical background (in the context of its treatment in philosophical analysis) of the idea of an obligation to be a persistent notion. We feel that the concept of an obligation

carries with it a default to be revoked (unless otherwise prevented) upon that obligation being discharged. Consider yourself in the position of being told by your superior at work that you were obliged (had) to attend a meeting. Suppose you discharged your obligation by attending, would you still continue to assume you were obliged to attend the meeting in question ? We think not !

Secondly, we believe that such an approach would unnecessarily clutter specifications and make them unmanageable. Imagine having to make sure that every time an obligation occurred that it also had to be revoked. It is by far easier to build in this default as part of the meaning of obligations, much in the spirit that a conventional frame assumption is employed in the conventional case of characterising state transforming actions.

Obligation sequences.

Above we made the observation that the intended effect of an obliged action could well be to introduce another obligation. We have called the generalisation resulting from this observation obligation sequences. We saw earlier the general form of obligation sequences; we can provide a more concrete example by a simple modification to axiom (15). Instead of the obliged action granting a permission, we can change it so that it now introduces another obligation:

$$\begin{aligned} \forall s \text{ Atport}(s, \text{Boston}) \rightarrow [\text{Sail}(s, \text{Dover})] \\ (\text{O}(\text{Unload}(s, \text{Timber})) \wedge [\text{Unload}(s, \text{Timber})] \text{O}(\text{Unload}(s, \text{Steel}))). \end{aligned}$$

This provides us with an example of an obligation sequence of length two; it reads as follows: "If a ship is docked at Boston and sails to Dover, it incurs an obligation to unload its cargo of timber as a result; but discharging this obligation leads to the introduction of another, namely to unload its cargo of steel". Taken in isolation we see that each individual obligation is still transitory in nature. However the recovery of the p-structure before the first obligation is incurred, is delayed until the entire sequence has been discharged. We saw general reasons for why this should be so when we were explaining our constraints on DAL models. All we need point out here, by way of summary, is that (accepting our arguments for obligations to be viewed as transitory interjections) it would seem unnatural (or at the very least counterintuitive) to want to recover the p-structures of scenarios in which obligations exist, since they are by construction only transitory. To reinforce this point consider the effect of discharging the last obligation, unloading steel, in the above example. If we recovered the p-structure

immediately preceeding this action we would be in the position of having to, once again, unload the ship's cargo of timber. Also, if, as in this case, the act of unloading timber does not change as a result of unloading steel, an obligation to unload steel will again occur. If we then reiterate our argument, it is easy to see that the shipping company as a whole will be forced to simply alternate between the action of unloading timber and steel.

Permission and Obligation Gaps.

The term gap is used in the philosophical logic literature, [Wri82] [FH81], to refer to the absence of information. We have already talked about incomplete information in various contexts; here we deal with two specific types, gaps in permissions and gaps in obligations.

In the case of permissions we have already seen how gaps or incompleteness may arise. Note for instance that the above specification gives us no direct information about when (under which conditions) ships may sail from Dover to Bombay. If we take a dynamic perspective it may be possible (because permissions are persistent) that a ship has permission (in a particular scenario) to sail from Dover to Bombay, but this is by no means gauranteed by the specification and may depend upon the particular model or implementation chosen. On the other hand, the specification does assure that the following holds:

$$* \quad \forall s \text{ [Sail}(s, \text{Bombay})] \text{ } \mathbf{n} \vee \neg \mathbf{n}.$$

This follows from the property that there are only two types of scenario, normative and non normative, represented by $\mathbf{n}$ and $\neg \mathbf{n}$ respectively. This in turn verifies the fact that "every action either leads to a normative or non normative scenario" is an axiom of DAL. This in turn means that this it is a property of every specification, and so, by necessitation, a property of every scenario of a particular specification; this means it must also be a consequence of every action in a specification. Hence, we can rewrite * as

$$\forall s \text{ ([Sail}(s, \text{Bombay})] \text{ } \mathbf{n}) \vee (\text{ [Sail}(s, \text{Bombay})] \neg \mathbf{n})$$

which, using the definition of permission in our axiomatisation of DAL, can be rewritten as:

$$** \quad \forall s \text{ P(Sail}(s, \text{Bombay})) \vee \neg \text{P(Sail}(s, \text{Bombay})).$$

Returning now to the originally observed gap, we see that

$$\forall s \text{ Atpport}(s, \text{Dover}) \rightarrow P(\text{Sail}(s, \text{Bombay})) \vee \neg P(\text{Sail}(s, \text{Bombay}))$$

is a special case of **. Therefore we can conclude that whenever a ship is docked at Dover, it either has permission to sail to Bombay or it does not have permission to sail to Bombay. All models will have this minimal property: some may decide (or fill the gap) by also having as a property one of the permitted disjuncts. The purpose of this illustration has been to demonstrate that specifications which have permission gaps (a special brand of underdeterminism on permissions) delay the issue of whether an action is allowed or not, either to a particular model of a specification (which may be a refinement) or to an actual implementation. In the case of our example the issue of whether (and in which contexts) ships are allowed to sail from Dover to Bombay will depend on further refinements of our specification or be left open to particular models.

What we have said in the case of permission gaps also holds in the case of obligation gaps. However, because of the dynamic aspects of obligations there is also another facet to the issue of obligation gaps. The heart of this rests around the notion of what it means for an action to *not* be obliged.

It should by now be clear that permissions are somehow more fundamental (mainly due to their persistence properties) than obligations. And as we have seen obligations are essentially temporarily imposed (if rather restricted) p-structures. On the other hand, the negation of an obligation (referred to as a non obligation from here on) representing the fact that some action is not obliged is persistent in the same way as permissions. Consider for instance our shipping company example; it is silent (has a gap) about the presence of obligations except in the case of ships sailing from Boston to Dover. This means that there exist valid models of this specification which introduce (by filling the gaps) new obligations not mentioned explicitly in the specification. Because of their rather global effects, this is not a very desirable phenomenon; we would really like to have better control on where obligations are incurred. We can achieve this to some extent by initialising the system in question (c.f. the discussion of a "birth" or initial system action in the previous chapter) to start off by having no obligation to do any action by asserting something along the lines of

$$[\text{Initial}] \forall \alpha \neg O(\alpha).$$

Subsequent actions (after this initial one) may then incur obligations with the advantage that the majority of non obligations (in fact all but the obliged action) are unaffected and

persist as does other information. We propose this strategy in order to keep tighter rein on where obligations might appear rather than a compulsory measure. It is of course quite easy to conceive of situations in which it is convenient to purposefully leave the prescription of obligations gappy. A specification in which a large number of refinement steps were needed in order to prescribe where obligations should be incurred is a good example.

We have just seen that non obligations share none of the transitory properties of their positive counterparts. Surely the next question to explore is, do they have any special static properties ? The only property that we thought may have some intuitive appeal can be summed up as follows: "Given that an action α is not obliged, it must also follow that there exists some action other than α that is permitted". Or, more succinctly:

$$i \quad \neg O(\alpha) \rightarrow \text{Pref}(\alpha).$$

Although this seems a plausible inference it is in fact incorrect; we only need to look at a scenario in which the following permission and obligation structure holds to find a counter intuitive example:

$$ii \quad P(\alpha) \wedge \neg \text{Pref}(\alpha) \wedge \neg O(\alpha)$$

The point to observe is that only having permission to do one action (and not anything else) does not imply that an action is obliged. This is due to the dynamic aspects of obligations which we have already discussed. Hence, expression (ii) is quite consistent; this would not be so if we accepted (i) as a general principle.

In summary then, non obligations are very different to their positive counterparts, both in static and dynamic terms. They should, we believe, be used to control the spurious occurrence of obligations by limiting the number of obligation gaps.

Deontic Reasoning.

Reasoning from specifications written in DAL, just like reasoning from SFOAL specifications, can be separated into two kinds: reasoning from a specification and reasoning from specific scenarios, within a specific model of a specification. In addition to the reasons we saw earlier, the latter type of reasoning is even more crucial if we are to reason about the dynamic aspects of obligations.

Before we proceed, the reader is asked to note that DAL is a proper conservative

extension of SFOAL and hence we retain the basic mechanism of proofs, as described in SFOAL, as a vehicle for correct reasoning in DAL. The major difference is that in addition to the structural properties of a specification, we now can also reason about the behavior of a system in terms of the concepts of permissible and obliged actions. Thus, questions such as, "After sailing to Bombay is the Discovery permitted to sail to Boston?" and "Which ships are obliged to unload timber at Dover?" can be answered.

To a large extent reasoning about deontic information is similar to reasoning about non deontic (or structural) information. For example, syntactically speaking, there is no real difference between the treatment of the property that a particular action is permitted and any other property consisting of a monadic predicate. We have, for instance, already seen that the permission operator conforms to all the proof manipulations (including the tautological forms) that any other monadic predicate does.

In addition to this we have the equivalence of $P(\alpha)$ with $[\alpha] \text{ } \neg$ in all normative scenarios. This means that we could, if we wished, avoid the introduction of a permission operator altogether, and reason about permitted actions in terms of whether or not the descriptions of these actions included the property that when performed they would result in normative scenarios. There are a number of reasons why we retain the permission operator. Firstly, it would be far more difficult, in large proofs, to manipulate action descriptions rather than expressions involving the (more concise) permission operator. Secondly, and perhaps more importantly, the introduction of a permission operator helps to distinguish clearly (our discrimination) between the structural (descriptive) and behavioral (prescriptive) aspects of actions. Put another way, we believe that a particular type of structural information, concerning whether (or not) an action reaches a normative scenario, is important and different enough to warrant syntactic discrimination. The difference between this type of coding (of concepts in existing primitive notions) and syntactic abstraction (the introduction, whether by definition, as in this case, or otherwise, of new syntax to capture the new concepts in a more direct way) has been fought on many battlefields including those of temporal and modal logics to name but two. In the end, it almost invariably amounts to a question of personal philosophy and taste.

As an illustration of reasoning about permissions consider the following assertion:

- i $[\text{Sail}(\text{Voyager}, \text{Boston})]$
- $[\text{Sail}(\text{Voyager}, \text{Dover})] \neg P(\text{Sail}(\text{Voyager}, \text{Boston}))$

We may show that this follows from the preceeding specification by the following trivial proof:

1. $\forall s \forall p \text{ [Sail}(s, p) \text{] Atport}(s) = p$ Asmp. Axiom (6)
2. $\text{[Sail(Voyager, Boston)] Atport(Voyager) = Boston}$ Subst. [Boston/p] 1
Subst. [Voyager/s] 1
3. $\forall s \text{ Atport}(s, \text{Boston}) \rightarrow$ Asmp. Axiom (15)
 $\text{[Sail}(s, \text{Dover}) \text{] } (O(\text{Unload}(s, \text{Timber})) \wedge [\text{Unload}(s, \text{Timber})] P(\text{Sail}(s, \text{Boston})))$.
4. $\text{Atport(Voyager, Boston)}$ Subst. [Voyager/s] 3
 $\rightarrow \text{[Sail(Voyager, Dover)] } (O(\text{Unload(Voyager, Timber)})$
 $\wedge [\text{Unload(Voyager, Timber)}] P(\text{Sail(Voyager, Boston)}))$.
5. $\text{Atport(Voyager, Boston)}$ 4 and SFOAL.
 $\rightarrow \text{[Sail(Voyager, Dover)] } O(\text{Unload(Voyager, Timber)})$
6. $\text{[Sail(Voyager, Boston)] [Sail(Voyager, Dover)] } O(\text{Unload(Voyager, Timber)})$
2, 5 and SFOAL.
7. $\text{[Sail(Voyager, Boston)] [Sail(Voyager, Dover)] } \neg P(\text{Sail(Voyager, Boston)})$
6 and O1.
Q. E. D.

The proof is trivial and should in itself need no further explanation; it does however help to raise an important question about reasoning with permissions. The point is that permissions in themselves do not allow us to reason about normative transitions; this is because our definition of the P operator is only a partial definition whose scope of definition is restricted to normative scenarios. Hence having the information $[\alpha] P(\beta)$ in a specific scenario s , does not mean that performing the action α will lead to a scenario in which performing β will lead to a normative scenario. To ensure this, we would need the additional property that α is permitted in the scenario s .

If we were interested in reasoning about normative transitions, this could be

accommodated by the introduction of a new modality defined in terms of actions and permissions, as follows:

$$\{\alpha\} \varphi \leftrightarrow (\mathbf{n} \wedge P(\alpha) \wedge [\alpha] \varphi).$$

Intuitively this new definition would read: "A normative α transition results in property φ if and only if the scenario in which it is performed is normative, the action α is itself permitted and, last but not least, the effect of performing α is φ ". Reasoning about normative transitions can be very useful in verifying that a specification ensures certain required permissible behaviour. It may for example help to highlight important deontic gaps which the specifier had inadvertently missed and need to be appropriately "filled". Again, note that the ability to reason hypothetically, by assuming that certain actions are (not) permitted can be very valuable for this kind of validation.

Reasoning about obligations can be divided into two kinds; reasoning about the static aspects of obligations and reasoning about the dynamic aspects.

The static aspects are straightforward to deal with; all our remarks about permissions also apply to obligations. Specifically the obligation operator may also be treated, in both manipulation and form, like a monadic predicate. In addition to this, we must also take into account the two axioms **O1** and **O2**. These axioms enforce all scenarios to have the p-structure,

$$P(\alpha) \wedge \neg \text{Pref}(\alpha),$$

whenever they have the property $O(\alpha)$ for any action α . In other words, the presence of an obligation in a scenario dictates (in an extreme way) which actions are (or are not) permitted in that scenario. We have just seen an example of this type.

The dynamic nature of obligations needs a little more explanation; at present we have only been able to semantically capture this aspect and so, as we have already seen, our logic is incomplete. Although completeness is often a desirable property for any logic to possess it should not be allowed to become a restricting barrier if it is not achieved. There are many examples of logics that are very useful in the analysis of applications and yet are not complete. We are thinking specifically about non monotonic logics which have general default rules, like the commonly adopted frame axiom, see for example [Rei80], [Jo80] and [GMS83].

The most common argument advanced by people who believe that a logic must

accommodated by the introduction of a new modality defined in terms of actions and permissions, as follows:

$$\{\alpha\} \varphi \leftrightarrow (\mathbf{n} \wedge P(\alpha) \wedge [\alpha] \varphi).$$

Intuitively this new definition would read: "A normative α transition results in property φ if and only if the scenario in which it is performed is normative, the action α is itself permitted and, last but not least, the effect of performing α is φ ". Reasoning about normative transitions can be very useful in verifying that a specification ensures certain required permissible behaviour. It may for example help to highlight important deontic gaps which the specifier had inadvertently missed and need to be appropriately "filled". Again, note that the ability to reason hypothetically, by assuming that certain actions are (not) permitted can be very valuable for this kind of validation.

Reasoning about obligations can be divided into two kinds; reasoning about the static aspects of obligations and reasoning about the dynamic aspects.

The static aspects are straightforward to deal with; all our remarks about permissions also apply to obligations. Specifically the obligation operator may also be treated, in both manipulation and form, like a monadic predicate. In addition to this, we must also take into account the two axioms **O1** and **O2**. These axioms enforce all scenarios to have the p-structure,

$$P(\alpha) \wedge \neg \text{Pref}(\alpha),$$

whenever they have the property $O(\alpha)$ for any action α . In other words, the presence of an obligation in a scenario dictates (in an extreme way) which actions are (or are not) permitted in that scenario. We have just seen an example of this type.

The dynamic nature of obligations needs a little more explanation; at present we have only been able to semantically capture this aspect and so, as we have already seen, our logic is incomplete. Although completeness is often a desirable property for any logic to possess it should not be allowed to become a restricting barrier if it is not achieved. There are many examples of logics that are very useful in the analysis of applications and yet are not complete. We are thinking specifically about non monotonic logics which have general default rules, like the commonly adopted frame axiom, see for example [Rei80], [Jo80] and [GMS83].

The most common argument advanced by people who believe that a logic must

necessarily be complete is that, if you cannot reason about some particular phenomenon of a formalism what is the point of retaining that phenomenon ? They would say that either the phenomenon has not been properly characterised (and should be changed so that completeness can be achieved) or else the phenomenon should be given up since it serves no realisable use in the logic.

Our own position on this, concerning DAL as a formalism for requirement specifications, is based on a distinction between the two roles (not often clearly distinguished) that a specification plays. Most people see a specification as an object which describes an intended system and which can be manipulated and reasoned about so that we can be sure that any concrete system that is built is correct. If this were the only way to view a specification the people who insist on completeness would have a case, but we believe that a specification is also a prescription to build a system. This is not such an extreme view given that reasoning about action transitions in most formal systems requires some form of implicit *model dependent* frame assumption. An example of this may be found in [Jo86] wherein the author also takes the approach that reasoning about actions is essentially a model based activity.

Having adopted this bifocal view of thinking that a specification should govern how a system is built as well as being the object against which the realised system is to be verified/validated, we can attempt to explain the incompleteness of our logic. In a nutshell we believe that the non axiomatisable features of our concept of obligation should not be discarded simply because we cannot accommodate them as part of our logical framework. Instead we can (and should) incorporate such features whenever appropriate as (prescriptive) constraints on the class of all correct system implementations. Hence, the reason for incorporating the dynamic nature of obligations as a collection of constraints on the models (and as a limiting case any implementations) of a specification. The constraints we impose are in a sense a particular form of frame condition which we could not (for reasons we have already mentioned) capture by rules in our logic. The constraints force all the models of a specification to exhibit the dynamic aspects of obligations, as we have already described them. We have already seen examples of how such reasoning may proceed and refer the interested reader to [CC87] for a more detailed treatment of how such model level reasoning can be achieved with the help of theorem proving tools.

necessarily be complete is that, if you cannot reason about some particular phenomenon of a formalism what is the point of retaining that phenomenon ? They would say that either the phenomenon has not been properly characterised (and should be changed so that completeness can be achieved) or else the phenomenon should be given up since it serves no realisable use in the logic.

Our own position on this, concerning DAL as a formalism for requirement specifications, is based on a distinction between the two roles (not often clearly distinguished) that a specification plays. Most people see a specification as an object which describes an intended system and which can be manipulated and reasoned about so that we can be sure that any concrete system that is built is correct. If this were the only way to view a specification the people who insist on completeness would have a case, but we believe that a specification is also a prescription to build a system. This is not such an extreme view given that reasoning about action transitions in most formal systems requires some form of implicit *model dependent* frame assumption. An example of this may be found in [Jo86] wherein the author also takes the approach that reasoning about actions is essentially a model based activity.

Having adopted this bifocal view of thinking that a specification should govern how a system is built as well as being the object against which the realised system is to be verified/validated, we can attempt to explain the incompleteness of our logic. In a nutshell we believe that the non axiomatisable features of our concept of obligation should not be discarded simply because we cannot accommodate them as part of our logical framework. Instead we can (and should) incorporate such features whenever appropriate as (prescriptive) constraints on the class of all correct system implementations. Hence, the reason for incorporating the dynamic nature of obligations as a collection of constraints on the models (and as a limiting case any implementations) of a specification. The constraints we impose are in a sense a particular form of frame condition which we could not (for reasons we have already mentioned) capture by rules in our logic. The constraints force all the models of a specification to exhibit the dynamic aspects of obligations, as we have already described them. We have already seen examples of how such reasoning may proceed and refer the interested reader to [CC87] for a more detailed treatment of how such model level reasoning can be achieved with the help of theorem proving tools.

3.5 Conclusions.

We hope that we have at least partially fulfilled some of the aims we motivated in the introduction to this chapter. As various remarks scattered throughout the sections of this chapter show, our work is by no means complete, either in the foundational sense or in the development of extensions to DAL. As a conclusion to this chapter we shall very briefly outline some current problems and propose some further extensions.

One loose end we have promised to remark on has to do with the Pref operator and the weakening of obligations. Currently, an obligatory action is one which must be performed next; this is sometimes too strong a constraint and it may be more appropriate to restrict the definition of obligation to be more local. One way to do this would be to define obligations relative to a smaller collection of actions so, given a set of actions 'A' and an obligatory action 'a', we may define $O(a \mid A)$ as the action 'a' having precedence over any action in the set 'A'. In general we may characterise such restrictions by corresponding changes in the definition of Pref; as an example, consider the following alteration:

$$\text{Pref}(\alpha_R) \leftrightarrow \exists \beta (\neg (\alpha = \beta) \wedge \neg R(\beta) \wedge P(\beta))$$

Where α_R denotes the set of actions having property R (note this set might be expressed as a more complex formula) and $\text{Pref}(\alpha_R)$ the property that we have a permissible alternative to all actions equivalent to α or having property R. Using this definition and retaining our usual axioms **O1** and **O2** for obligation we get:

$$O(\alpha) \rightarrow P(\alpha)$$

$$O(\alpha) \rightarrow \forall \beta (P(\beta) \rightarrow (\alpha = \beta) \vee R(\beta)).$$

Hence obligations are not restricted to permitting only one action but also allow for the fact that any action with the property R may also be permitted. Although we shall not explore this avenue further, it should not be hard to see how this weaker notion of obligation admits the possibility of more than one obligation at a time. Also, given a structured action category it is not difficult to see how complex obligation structures may be set up.

There are two further ways in which we make our current characterisation of obligations a little less strong; one involves the introduction of deontic operators which deal with permissible and obligatory states of affairs and the other is to relativise the concepts of

permitted and obligatory actions to named performers of those actions.

The former would provide us with a less fine (more abstract) means of permitting and obligating actions by allowing us to permit or oblige the result of an action without naming the specific act. Given that many actions may result in the same permitted/obliged property, it is not difficult to see the advantages of such operators. We feel that they would be particularly useful when we wished to permit/oblige a class of actions - based on a common effect - without being forced to decide a particular one. As an illustration of this, consider the requirement that the shipping company in our earlier example is permitted (in some suitably described situation) to ship tea from Boston to Bombay. Given that we do not require a particular action to be permitted - just an effect, there may be many permissible actions which will have the desired effect. It could be, for instance, that two or even all three ships are at Boston, loaded with tea and with permission to sail to Bombay. In the next chapter we shall see how DAL can be extended to accomodate such operators.

The shipping example of the previous section also serves to highlight another major deficiency of DAL; in order to see this consider what happens when the Voyager incurs an obligation to unload its timber. One quite obvious consequence is dictated by the obligation axioms **O1** and **O2** which require that permission for all other actions be suspended. This means, amongst other things, that the Discovery is not allowed to sail whilst the Voyager is unloading timber. Clearly this is not what happens in reality; it is much more reasonable to think of each ship as having more autonomy in operation. One way to create this more local specification of behaviour is to introduce the idea of a system being composed of a number of active agents each having their own permissible and obligatory acts. We shall see how this can be achieved in the chapter after next, where we will also need to extend the action description machinery of our logic in order to be able to talk about specific agents doing actions.

CHAPTER FOUR

ACTION COMBINATORS

4.1 Introduction.

So far we have restricted ourselves to dealing solely with simple or primitive actions. Reality suggests that some actions are of a more complex structure and their description may rely on combining, in various ways, the descriptions of the simpler actions of which they are built. Modifying an employee's overtime rate in a company database is an example of a complicated action; it will, quite probably, involve a number of simpler actions, such as, searching for the particular employee record, accessing the appropriate information and then altering it. These actions may in turn be analysed in terms of yet simpler actions; for instance, altering the overtime rate may involve the action of writing the contents of a machine register to a secondary storage device such as optical disk, which in turn may involve actions to move disk heads, which and so on. As we can see, if we get too carried away, the process, of analysing complex actions in terms of simpler ones, can be continued *ad infinitum* .

In order to avoid this infinite regress the simplicity or atomicity (in the sense of being primitive and not analysable in terms of yet simpler actions) of certain actions has to be accepted by everyone, since the only alternative would be to say that every action, and hence also its description, must be infinitely complex or consist of infinitely many distinguishable acts, and this would be unworkable if not absurd. The only question then, is which actions are to be regarded as atomic ? Attempting to answer this in the abstract is rather like aiming at a moving target.

As far as requirement specifications go, the class of actions considered to be primitive would depend both at the level at which the requirements are specified and the nature of the system (or application domain). The more familiar domain of programming languages provides us with a good analogy with which to illustrate this, take for instance the programming language Pascal [Wir71], which has a number of primitive actions such as assignment and procedure call. These actions are in themselves fairly complex and can be interpreted in terms of register and memory transfers but as far as Pascal is concerned they are regarded as primitive. However if we were to look at lower level languages such as assembly code, actions like register transfers may in fact be taken as primitive. The key point here is that for a given program P we can analyse P in terms of two different sets of primitive actions, depending on the level at which we wish to do so. This also holds true for specifications, given a system S we may, according to how abstract we wish our specification of S to be, employ different collections of primitive

actions.

Quite apart from considering the level at which we construct system specifications the system itself also affects what class of actions should be considered primitive. Returning to our programming languages analogy, it would be inappropriate to use an imperative language like Pascal when we are interested in writing a program involving a large amount of functional recursion; Hope [BMS80] may be a better choice because it provides a more appropriate collection of primitive actions. It should be obvious that this idea of an appropriate class of primitive actions is also relevant to a systems specification and is to a certain extent determined by the level at which a system is analysed.

The analogy does not end here; once an appropriate collection of primitive actions has been identified a programming language must also provide the means by which they can be combined in order to construct programs. The common devices for this are the branching, looping and sequencing of actions; programs constructed in this way can be thought of as higher level actions interpreted in terms of executing specific combinations of simpler actions. This idea of allowing actions (at the same level) to be combined in order to form more complicated actions is also very important in the area of specification. There are several reasons for this; consider the point about action refinements. If we are to interpret a complex action in terms of more simplistic ones it will usually be in terms of a particular combination of these. For example the assignment action of a pascal program may be refined into a sequence of actions involving memory and register transfers and a while loop may be interpreted as a sequence involving a test action followed by either execution of the loop or something else. Without the ability to express an ordered sequence of actions or a branch (possibly involving a jump action) at the machine code level it would not be possible to refine the higher level pascal actions. For specifications (as for programming languages) this would mean that we could not reduce the complexity of an action by refining it into a specific combination of a number of simpler actions, all that would be possible would be the translation of one action into another action. As we explained in chapter two and the introduction to this thesis this process of refining specifications is crucial if we are to arrive at well implemented operational systems and so therefore is the adoption of mechanisms to combine actions.

Apart from their role in relating different levels of specification refinement, by interpreting complex actions in terms of collections of simpler ones, mechanisms which combine actions (referred to from here on as action combinators) also provide an invaluable structuring tool. It may for instance be convenient to talk explicitly about a

particular combination of actions, at the same level of specification, rather than hiding this structure under some, as yet to be interpreted, monolithic action. Database transactions, [LL81] and [Bar81], are examples of this; a transaction is considered to be a specific indivisible (non interruptible) combination of database operations (updates) where the successful execution of the transaction relies on the successful execution of the actions it comprises. Apart from their atomicity property database transactions are also useful as a means by which valid (permitted/desirable) combinations of database operations may be organised. Because of this many database specification languages incorporate some means of combining actions in this way; the interested reader is referred to [LEG85] and [KMS86] where this approach has been adopted in the algebraic and logical (respectively) settings. The success of transactions as a modelling tool for database construction and design led very quickly to the idea being used by related fields such as distributed systems (networks); see for example [He84] and even programming languages see for instance [Pra83]. Action combinators have also been used in the more general arena of programming logics; here the idea seems to be to model various programming language constructs on an a small number of basic action combinators. This is exactly the approach taken in [Gol82] wherein the author uses basic action combinators such as sequential composition, arbitrary iteration and non deterministic choice to describe programming constructs such as "If-then-else" and "While-do".

Our interest in action combinators stems from a simple observation of reality, as does for example the concept of transactions; actions just like other less dynamic objects possess structure. We had already hinted at this in chapter two, where we saw how we could ascribe properties to and relationships between action terms (names). We also saw how to construct functions able to accept as argument and return as result actions; the particular action combinators we have in mind are of this functional form. More specifically they are logical binary functions which accept as argument two actions and return a third, the idea being that the resulting action is a combination of the other two. The particular action combinators we have chosen are the common ones of sequential composition ($;$), parallel composition ($\parallel$) and non deterministic choice ($+$). It should be noted that other combinators could be defined including different interpretations of these but we feel that these three are basic enough to illustrate the points we wish to make. These logical functions will usually be written in infix notation like $\alpha;\beta$ and may be recursively applied. This last point is an important feature of our system and rests on the property that actions formed using action combinators are just that they are actions and so have the same rights as primitive actions to be combined with other actions.

A major difference in our approach to action combinators and others, see for example [Gol82], is that we do not consider a combination of actions at an operational level - to be thought of as one action being done in parallel with another for example - but rather at the more logical level of how the effects of specific combinations of actions are consequences of the effects of the actions forming that combination. In the next section we will see how DAL as a logic can be extended to accommodate this view of action combinators.

So far we have talked about combinators as a means for sticking together different actions with the aim of forming more complex ones; from here on we shall refer to these as primary action combinators. As we shall see in the next section actions formed using primary action combinators will have to be viewed in two ways: firstly at a structural level, concerned with the effects of performing a combined action in terms of the effects of its constituent actions and secondly, at a more prescriptive level concerned with the permissibility or obligatoriness of the action in terms of the permissibility or obligatoriness of its constituent actions.

If we mix these two levels of structure and prescription another useful class of operators which we shall call secondary action combinators can be formulated. Imagine for a moment that we are not interested in which particular action is permitted or obliged to occur just in what is permitted or obliged to be the case. Consider for example the statement "It is obligatory that message M be delivered to node N"; here we are not interested in which action (or combination of actions or sequence of actions) is employed to deliver the message M, only that it be delivered. However, if we are to be realistic we must assume that the message will be delivered by an action, either primitive or complex, or a sequence of actions available to the system. It is on the basis of this assumption that we build operators which allow us to express when certain properties (as opposed to actions) are either permitted or obliged.

Intuitively a property ϕ is permitted (or obliged) if there exists at least one action which when performed leads to a scenario in which ϕ holds and is permitted (obliged). To distinguish between permission / obligation for properties and permission / obligation for actions we name the former as Per and Obl. These operators are intended to denote those properties which are permissible or obligatory in terms of single actions, either primitive or complex. In addition to these we define two further operators Sper and Sobl which are used to assert which properties are permissible or obligatory via the performance of either a permissible or obligatory sequence of actions (primitive or

complex). As such Per and Obl are special cases (essentially the single action sequences) of the Sper and Sobl operators, we shall see why we have singled out the singleton action sequence for special treatment later. It is because all these operators refer only to anonymous actions or sequences there of that we have called them secondary action combinators.

Secondary action combinators can to some extent be compared to the more conventional operators of temporal and modal systems. For example the "Next" operator, usually represented by the symbol "O" (see [RU71] or [Pnu77] for more about this and other temporal operators) which asserts that a certain property must hold in the next "state" of the system or computation corresponds to our Obl operator. The major difference, and we believe advantage, between either conventional temporal or modal systems and ours is that we tie our operators explicitly to the actions available to the system we are trying to specify. To see an example of this consider the following two expressions

$O(\varphi)$ and $Obl(\varphi)$

the first is read "In the next observable state of the system the property φ holds" and the second as "There exists some action, either primitive or complex, which when performed leads to a scenario in which φ holds and further that action is obliged (must) happen. Both expressions are similar in that in the next system state φ holds and no specific action is mentioned in either expression but they differ in one very important way: the Obl operator is defined explicitly in terms of the existence of a suitable system action which is obliged to happen, the temporal operator on the other hand says nothing about how the next state is reached and can only at best be given an implicit relationship to system actions.

However, there are advantages to be had from not making an exclusive tie between system actions and temporal operators such as "Next"; one major advantage is that one can define operators that can be used to talk about a system's past states as well as potential future states. Because our secondary action combinators are based on actions we can only deal with potential futures, the extent of inconvenience this brings depends mostly on personal taste. A temporal extension of the framework described here, which address this problem can be found in [KM88] where temporal operators which talk about a system's past states (scenarios) are introduced; we shall return to this point in the concluding chapter of this dissertation.

The remainder of this chapter is structured as follows: in the next section we shall

introduce and discuss the logical apparatus to incorporate the primary action combinators; this is followed by a section on secondary action combinators. Section four summarizes the logical extensions of sections two and three naming the extended logic as CDAL finally, in the fifth section, we discuss an electronic mail network example written in CDAL and conclude with some general observations and comparisons.

4.2 Primary Action Combinators.

In this section we shall introduce three action combinators: sequential composition, parallel composition and non deterministic choice. The purpose of these combinators is to allow us to construct actions which are specific combinations of other actions. The conventional, and perhaps at first more intuitive, approach to expressing such actions has been a very operational one, the idea being that actions which are combinations of other actions are ones which depend upon performing their constituent actions in some particular order. For example a parallel action may be dependent on a particular model of concurrency. But since we are in the business of specifying rather than constructing actions, we should perhaps think of action combinators as a means of *describing* the effects of some actions based on the descriptions of others without recourse to how such actions are constructed/implemented. The remainder of this section is concerned with an explanation and justification of this view. From here on we shall use the term "primitive" to refer to those actions which do not involve the use of primary action combinators and "complex" for those actions which do.

In order to incorporate action combinators into our framework we will need some new logical syntax and some new axioms, in short we must extend DAL. Again, rather than rewrite the logic of DAL we shall only focus on the extension, the reader is reminded that the language and axioms that follow are intended to augment the language and axioms of DAL.

Language extension.

DAL: All the syntax and formation rules of DAL,

plus the following logical functions:

Functions :

; of type $\langle \text{Act}, \text{Act}, \text{Act} \rangle$

|| of type $\langle \text{Act}, \text{Act}, \text{Act} \rangle$

+ of type $\langle \text{Act}, \text{Act}, \text{Act} \rangle$

Axioms.

DAL: All the axioms of DAL *plus* the six below.

- (1) $P(\alpha;\beta) \rightarrow ([\alpha;\beta] \phi \leftrightarrow [\alpha][\beta] \phi)$
- (2) $P(\alpha \parallel \beta) \rightarrow ([\alpha] \phi \wedge [\beta] \psi \rightarrow ([\alpha \parallel \beta] \phi \wedge \psi))$
- (3) $\alpha \parallel \beta =_{\text{Act}} \beta \parallel \alpha$
- (4) $P(\alpha+\beta) \rightarrow ([\alpha] \phi \wedge [\beta] \psi \rightarrow ([\alpha+\beta] \phi \vee \psi))$
- (5) $([\alpha] \phi \wedge [\beta] \psi \wedge \neg [\alpha] \psi \wedge \neg [\beta] \phi) \rightarrow (\neg [\alpha+\beta] \phi \wedge \psi)$
- (6) $\alpha+\beta =_{\text{Act}} \beta+\alpha$

Note that we are still using ϕ and ψ as meta variables that range over formulae and the symbols α and β to denote (primitive or complex) actions. The remainder of this section is in two parts: the first looks at the structural properties of the combinators whilst the second examines their normative properties. We shall delay the treatment of models for specifications written in this extended logic until the section where we talk about such specifications.

4.2.1 Structural aspects

Since all three action combinators are binary logical functions which take two actions as argument and return an *action*, as a result all the axioms and formation rules that apply to primitive actions also apply to complex ones. This is because all complex actions are still members of the action category **Act**. Specifically, this means that complex actions may be described using the one place modal connectives " $[_]$ ", subject of course to the same axiomatic constraints placed on primitive ones. For example $[\delta] \text{TRUE}$ and $([\delta] \phi \wedge [\delta] \psi) \rightarrow ([\delta] \phi \wedge \psi)$ are valid for all complex actions δ . But in addition to this, a complex action's description does to some extent also depend upon the descriptions of the actions from which it is composed. It is these latter dependencies that axioms one to six of this section are intended to capture.

In order to simplify the following discussion we shall assume the leftmost/outermost antecedent of those axioms which govern how a complex action's description depends on the descriptions of its constituent actions, and focus only on the detached consequents. For example in the case of sequential composition the equivalence

$$1.1 \quad [\alpha;\beta] \phi \leftrightarrow [\alpha][\beta] \phi$$

is only logically necessary when the action $\alpha;\beta$ is permitted. Similarly, the axiom characterising parallel composition may be stripped of its leftmost and outermost antecedent so that we have

$$2.1 \quad ([\alpha] \phi \wedge [\beta] \psi) \rightarrow ([\alpha \parallel \beta] \phi \wedge \psi)$$

This simplification can also be made in the nondeterministic case resulting in the weaker axioms

$$4.1 \quad ([\alpha] \phi \wedge [\beta] \psi) \rightarrow ([\alpha+\beta] \phi \vee \psi)$$

These weaker axioms together with axioms three, five and six of the original set will be sufficient to describe most of the structural nature of complex actions; when we talk about the normative aspects of action of complex actions we will see why we need the original stronger versions of 1.1, 2.1 and 4.1.

Sequential composition.

The sequential composition operator is used to describe the effects of an action composed of two actions which are performed in sequence. This is precisely what axiom 1.1 is intended to characterise; the substance of this axiom is that it equates the structural effects of a sequentially composed action, $\alpha;\beta$ say, with the effects of performing the action β after performing the action α . The question which immediately comes to mind is, if doing a sequentially composed action is equivalent to doing the actions in sequence, why introduce a special combinator? This point may be answered by looking at the atomicity of sequentially composed actions. The action named $\alpha;\beta$ is to be viewed as just that, **an** action, and not a sequence of actions.

Perhaps an example will help to reinforce this difference: assume the existence of two actions, "Hire-Sam" and "Fire-Sam" representing database updates in a company's personnel department. Further consider the descriptions of these actions to be

$$S1 \quad [\text{Hire-Sam}] (\text{Employed}(\text{Sam}) \wedge \text{Scale}(\text{Sam}) = 1)$$

$$S2 \quad \text{Scale}(\text{Sam}) = 1 \rightarrow [\text{Promote-Sam}] \text{Scale}(\text{Sam}) = 2$$

with the obvious interpretation that hiring Sam registers him as being employed and places him on the first pay scale and promoting him raises him up to the next pay scale. Then, according to axiom 1.1 the description of the action "Hire-Sam;Promote-Sam" can be expressed as

$$(i) \quad [\text{Hire-Sam;Promote-Sam}] \phi \leftrightarrow [\text{Hire-Sam}][\text{Promote-Sam}] \phi$$

which, ignoring information other than Sam's pay scale status and noting that Sam can have only one pay scale (since we use a function to model this information), means that

we have

$$(ii) \quad [Hire-Sam;Promote-Sam] \text{ Scale}(Sam) = 2.$$

We can show this by the following proof:

- | | | |
|------|--|-----------------------|
| 1). | $[\alpha;\beta] \varphi \leftrightarrow [\alpha][\beta] \varphi$ | Axiom 1.1 |
| 2). | $[Hire-Sam;Promote-Sam] \varphi \leftrightarrow [Hire-Sam][Promote-Sam] \varphi$ | Universal Inst. |
| 3). | $[Hire-Sam] (\text{Employed}(Sam) \wedge \text{Scale}(Sam) = 1)$ | Assumption S1 |
| 4). | $\text{Scale}(Sam) = 1 \rightarrow [Promote-Sam] \text{Scale}(Sam) = 2$ | Assumption S2 |
| 5). | $[Hire-Sam] \text{Employed}(Sam) \wedge [Hire-Sam] \text{Scale}(Sam) = 1$ | SFOAL and 3). |
| 6). | $[Hire-Sam] \text{Scale}(Sam) = 1$ | $\wedge$ elim and 5). |
| 7). | $[Hire-Sam] (\text{Scale}(Sam) = 1 \rightarrow [Promote-Sam] \text{Scale}(Sam) = 2)$ | Necessitation, 4 |
| 8). | $([Hire-Sam] \text{Scale}(Sam) = 1)$ | |
| | $\rightarrow ([Hire-Sam] ([Promote-Sam] \text{Scale}(Sam) = 2))$ | SFOAL and 7). |
| 9). | $[Hire-Sam][Promote-Sam] \text{Scale}(Sam) = 2$ | M.P. 6) and 8). |
| 10). | $[Hire-Sam;Promote-Sam] \text{Scale}(Sam) = 2$ | 2) and 9). |

Q. E. D.

We are now in a position see how the sequentially composed action "Hire-Sam;Promote-Sam" differs from performing the action Hire-Sam followed by performing the action " Promote-Sam".

Assuming that Sam is not currently employed by the company in question then performing the database update of hiring Sam will result in a database state (system scenario) which if queried will tell us that Sam is employed and that he is on the first pay scale. Some time later Sam has shown his worth so we decide to promote him and in

order to reflect this decision in the company database we perform the update "Promote-Sam". When we query the new database state we will find that Sam has been placed on the second level pay scale. Now, contrast this update sequence of hiring and promoting Sam with performing under the same assumption - that Sam is not an employee of the company - the sequentially composed action "Hire-Sam;Promote-Sam". All we can be certain about the latter is that it will lead to a database scenario in which Sam is on the second level pay scale, this means that at no time (according to the company database) is Sam on the first pay scale. The reason for this is that performing the action "Hire-Sam;Promote-Sam", although structurally equivalent to first performing "Hire-Sam" and then performing "Promote-Sam", is an indivisible action whose effects can only be realised when the whole action terminates. So although the effects of sequentially composed actions are equivalent to the corresponding sequences of actions they are not operationally equivalent.

It is not difficult to find other examples of where the indivisibility feature of this combinator is useful. Concurrent systems are a case in point; here it is often essential that a sequence of actions is not interrupted; the success of performing the sequence relies on the atomicity (or indivisibility) of the sequence. Our sequential composition combinator can be used to describe such action sequences. The generic name given to sequences of this nature is "transaction", a term arising from the area of data and knowledge bases [Bar81] and [LL81], but applied as a principle in many fields including distributed/concurrent computing [He84].

A further consequence of axiom 1.1 is that sequentially composed actions are associative; consider the sequentially composed action $\alpha;\beta$ sequentially composed with the action δ . This may be written $(\alpha;\beta);\delta$ and read as "the action $\alpha;\beta$ sequentially composed with the action δ ", but it may just as well be written " $\alpha;(\beta;\delta)$ " and read as the action " α " sequentially composed with the action " $\beta;\delta$ ". We can see this by expanding both forms into their corresponding iterated modalities

$$[\alpha;(\beta;\delta)] \varphi \leftrightarrow [\alpha][\beta;\delta] \varphi \leftrightarrow [\alpha][\beta][\delta] \varphi$$

$$[(\alpha;\beta);\delta] \varphi \leftrightarrow [\alpha;\beta][\delta] \varphi \leftrightarrow [\alpha][\beta][\delta] \varphi$$

Sometimes, where it is clear from the context, we may drop brackets altogether and write actions of the form $\alpha;\beta;\delta$. Unlike the other two combinators sequential composition is not symmetric; the action $\alpha;\beta$ is quite different from the action $\beta;\alpha$, again, we can see this by writing out their equivalent forms:

$$[\alpha;\beta] \phi \leftrightarrow [\alpha][\beta] \phi$$

$$[\beta;\alpha] \phi \leftrightarrow [\beta][\alpha] \phi$$

and so because (in general) performing the action β after performing the action α is not the same as first performing β and then performing α neither are the corresponding sequential compositions.

Parallel composition.

Whereas the sequential composition combinator allows us to describe actions which are structurally equivalent to actions performed in certain (ordered) sequences the parallel composition combinator allows us to describe actions that produce the effect of two actions in parallel. Intuitively the idea is that the effect of a parallel action, written $\alpha \parallel \beta$, is the combined effects of the individual actions α and β . Axiom 2.1 is intended to capture this:

$$2.1 \quad ([\alpha] \phi \wedge [\beta] \psi) \rightarrow ([\alpha \parallel \beta] \phi \wedge \psi).$$

It is read as "if the effects of the action α and β are ϕ and ψ respectively, then the effect of $\alpha \parallel \beta$ is both ϕ and ψ ". It should be noted that this characterisation is only valid under what has become known as the principle of non interference. This principle states that the effects of the actions from which the parallel action is constructed do not interfere with each other. The consequences of interfering actions range from inconsistent results at worst to unpredictable behaviour at best. In our present framework both these types of interference are modelled as leading to inconsistent scenarios; we can best illustrate this with a few examples.

The worst effect of action interference is inconsistency; we can see an example of this by extending our company database example with the action of firing Sam; both the hiring and firing actions may then be described as

$$[\text{Hire-Sam}] \text{Employed}(\text{Sam})$$

$$[\text{Fire-Sam}] \neg \text{Employed}(\text{Sam})$$

which if combined in parallel would result, according to axiom 2.1, in the action description

$$[\text{Hire-Sam} \parallel \text{Fire-Sam}] (\text{Employed}(\text{Sam}) \wedge \neg \text{Employed}(\text{Sam}))$$

which would, if performed, lead to an inconsistent scenario, since Sam cannot be

employed and unemployed at the same time.

Parallel actions may also interfere with each other depending on how the parallelism is to be achieved, given the following action descriptions

$$\forall x, y \forall e \text{ Salary}(e) = y \rightarrow [\text{Raise}(e, x)] \text{ Salary}(e) = y + x$$

$$\forall x \forall e (\text{Salary}(e) = x \wedge x > 10,000) \rightarrow [\text{Bonus}(e)] \text{ Salary}(e) = x + 50$$

$$\forall x \forall e (\text{Salary}(e) = x \wedge x \leq 10,000) \rightarrow [\text{Bonus}(e)] \text{ Salary}(e) = x + 100$$

for raising an employee's salary and for giving him a bonus, how would we interpret the action

$$\text{Raise}(\text{Sam}, 1000) \parallel \text{Bonus}(\text{Sam}) ?$$

which attempts to both raise the salary of the employee Sam and give him a bonus. There is of course no single way of interpreting the physical effects which this action will have when performed against the company database. As we can see from the action descriptions of the individual actions the effect of giving an employee a bonus is dependent upon the current value of his/her salary. So increasing someone's salary and trying to evaluate their bonus at the same time raises some problems; do we want the bonus to be evaluated according to an employee's present salary or after his raise ? Perhaps it would have been better to use sequential rather than parallel composition here, forcing a particular order.

Parallel actions that interfere in this way (essentially through the shared value of an employee's salary) are not adequately treated in our formalism because we do not provide a means to describe how parallelism should take place. Instead we try to capture the effect of a parallel action purely in terms of the effects of the actions from which it is composed; this means that some parallel actions (those which depend on how parallelism is achieved) may lead to inconsistent scenarios. We can see an instance of this by applying axiom 2.1 to the above actions, assuming Sam has a current salary of 9,200 we can see that the following descriptions hold:

$$[\text{Raise}(\text{Sam}, 1000)] \text{ Salary}(\text{Sam}) = 10,200$$

$$[\text{Bonus}(\text{Sam})] \text{ Salary}(\text{Sam}) = 9,300$$

combining these in parallel we obtain

$$[\text{Raise}(\text{Sam}, 1000) \parallel \text{Bonus}(\text{Sam})] (\text{Salary}(\text{Sam}) = 10,200 \wedge \text{Salary}(\text{Sam}) = 9,300)$$

which describes an action that results in an inconsistent scenario. We hope that these examples illustrate why, at the logical level of describing parallel actions, the principle of non interference is an important one. We shall return to this principle when we talk about specifications written in this extended logic.

The intuitive idea of a parallel action may be summarized as follows: the effect of a parallel action is equivalent to the logical conjunction of the effects of the individual actions from which it is composed. However, the reverse does not in general hold, if it did axiom 2.1 would be an equivalence and this would mean accepting the following as an axiom:

$$2.2 \quad [\alpha \parallel \beta] \phi \wedge \psi \rightarrow [\alpha] \phi \wedge [\beta] \psi$$

We can see why 2.2 is not acceptable by observing what "happens" when we parallel compose two actions: given $[\alpha] \phi$ and $[\beta] \psi$ axiom 2.1 forces $[\alpha \parallel \beta] \phi \wedge \psi$, but in so doing loses information about which action contributed which effect. Given an arbitrary parallel action whose description is $[\alpha \parallel \beta] \phi$ it is not in general possible to ascertain what are the individual effects of α and β . This is because we describe actions in terms of how they change scenarios and not how they change our representation of scenarios; the reader is reminded about our discussion about theories and theory presentations in chapter two. All we need note here is that a parallel action need not result in a scenario in which the combined effects of its constituent actions are represented as a simple logical conjunction but may lead to a less direct syntactic representation of these conjunctive effects.

A secondary argument as to why 2.2 is unacceptable can be given by recognising the need to describe the effects of some parallel actions as more than the minimal conjunction of effects, as imposed by axiom 2.1, of the individual actions. As an example of this phenomenon consider an enterprise whose business is the buying and selling of shares and bonds. The following two action descriptions capture the effects of buying shares and bonds respectively,

$$\forall x, y \forall c \text{ Shares}(c) = y \rightarrow [\text{Buy-Shares}(c, x)] \text{ Shares}(c) = y + x$$

$$\forall x, y \forall c \text{ Bonds}(c) = y \rightarrow [\text{Buy-Bonds}(c, x)] \text{ Bonds}(c) = y + x$$

Here, the variable 'c' denotes from which company the shares or bonds are being bought, 'x' denotes the quantity of shares or bonds being purchased and the function 'Share' represents the number of shares or bonds of company 'c' currently held. The action descriptions themselves should be self explanatory. From 2.1 we know that the

parallel composition of these two actions can be described as

$$\begin{aligned} & \forall w, x, y, z \ \forall c, c' \ (\text{Shares}(c) = w \wedge \text{Bonds}(c') = y) \\ & \rightarrow [\text{Buy-Shares}(c, x) \parallel \text{Buy-Bonds}(c', z)] \\ & \quad ((\text{Shares}(c) = w + x) \wedge (\text{Bonds}(c') = y + z)). \end{aligned}$$

Now assume that the Monopolies Commission imposes restrictions on our business so that if we were to buy over five hundred shares and bonds of the same company we have in turn to sell one hundred of our own corporate share issue. We can represent this formally as

$$\begin{aligned} & \forall x, y \ \forall c \ (y \leq x \wedge y > 500) \rightarrow \\ & \quad [\text{Buy-Bonds}(c, y) \parallel \text{Buy-Shares}(c, x)] O(\text{Sell-Shares}(\text{Corporate}, 100)). \end{aligned}$$

This additional action description adds to the general information we have about buying shares and bonds in parallel and is only possible because the effects of a parallel composition are not necessarily equivalent to just the effects of its constituent actions. As long as we are careful not to contradict the logical meaning, as stated by axiom 2.1, of parallel compositions we may in this way increase the information we have about parallel compositions. Although we have not given an action description for the action of 'Sell-Shares' it should not be too difficult to see that an obligation to sell the business's own corporate shares will not contradict the effects of buying another company's shares and bonds. It is not hard to visualize other examples of this kind; we shall call parallel actions of this kind non minimal parallel compositions. We will call parallel actions which at all times exhibit *only* the conjunction of the effects of their component actions minimal parallel compositions.

As we hinted in our discussion of the non interference principle our characterisation of parallel actions allows us to delay (or rather abstracts from) making any decisions about how a future implementation will perform these parallel actions. There are a number of choices; we may for example choose true parallelism and run the individual actions concerned concurrently (or at least with some overlap), we may choose to arbitrarily serialize them and so only give the appearance of parallelism or we may employ a combination of these. It is important to distinguish our notion of parallel combinator from superficially similar notions in specification formalisms like CCS [Mi80] and Petri nets [Pe79]. Unlike these other formalisms we do not intend parallelism to have a meaning based on a particular model of concurrency; our aim has been to describe the effects of a parallel action in terms of other actions independently of any specific model

of concurrency. We believe that initial specifications should not be biased with particular views of how parallelism is to be achieved, but should admit many potential refinements in various models of concurrency, whether it be the interleaving models of CCS type systems or the truly concurrent Petri net models. The sacrifice that we make for this independence is that we are restricted by the principle of non interference. Which means that unlike CCS we do not model behaviour based on interfering actions, such as our salary increase and bonus award examples, where it depends on which action (increase in salary or the awarding of a bonus) is performed first. CCS can admit this kind of interference because the meaning of its parallel composition combinator is based on an interleaving model of concurrency which means that a parallel action is reduced to an arbitrary (order of) execution of its constituent actions. We shall return to how we extend our framework to incorporate models of concurrency in the concluding chapter of this work, after introducing the idea of an agent.

The parallel composition combinator is not in general associative, this is a direct result of allowing non minimal parallel compositions. We can see this by expanding the two action forms " $\alpha \parallel (\beta \parallel \delta)$ " and " $(\alpha \parallel \beta) \parallel \delta$ " and showing them not, in general, to be equivalent. Again, in symbols our task is to show that

$$E1. [\alpha \parallel (\beta \parallel \delta)] \varphi \leftrightarrow [(\alpha \parallel \beta) \parallel \delta] \varphi$$

is not valid. Assuming that the action descriptions of the individual actions α , β and δ are

$$[\alpha] \psi, [\beta] \gamma \text{ and } [\delta] \tau$$

then the left hand side of our, as yet, hypothesized inequality can be expanded as follows, assume that the action description of the action " $\beta \parallel \delta$ " is $[\beta \parallel \delta] \pi$ then by axiom 2.1 we have

$$[\alpha \parallel (\beta \parallel \delta)] (\psi \wedge \pi)$$

now because of non minimal parallel compositions π may logically subsume the combined effects of the actions β and δ , if this were the case we could then write

$$[\alpha \parallel (\beta \parallel \delta)] (\psi \wedge \gamma \wedge \tau \wedge ?)$$

where ? denotes some effect over and above, and as a result of parallel composing β and δ , than that of the necessary minimal. Looking now at the right hand side of E1 and following a similar expansion we obtain

$$[(\alpha \parallel \beta) \parallel \delta] (\pi' \wedge \tau)$$

where π' represents the effect of the parallel action " $\alpha \parallel \beta$ ". Again because of non minimal compositions the effect of performing " $\alpha \parallel \beta$ " may logically subsume the conjunction of the individual effects of the actions α and β . As previously we may indicate this by writing

$$[(\alpha \parallel \beta) \parallel \delta] (\varphi \wedge \gamma \wedge ?? \wedge \tau)$$

it is not difficult to see that the two expansions differ only in the unknown effects of non minimally parallel composing α and β and non minimally parallel composing β and δ . Since these unknown quantities are going to be specification specific they are not (in general) going to be the same and so parallel composition is not going to be associative. However the above also shows that all actions which are minimally parallel composed are in fact parallel associative. We can see this by noting that there will be no effects, special to particular parallel compositions of actions, other than those of the constituent actions and hence no unknown effects in the above expansions.

Unlike sequential composition parallel composition is symmetric, in the case of minimal parallel compositions we can see this by considering the following two arbitrary action descriptions

$$[\alpha] \varphi \quad \text{and} \quad [\beta] \psi$$

and their two associated parallel compositions which according to axiom 2 have the following action descriptions

$$[\alpha \parallel \beta] \varphi \wedge \psi \quad \text{and} \quad [\beta \parallel \alpha] \psi \wedge \varphi$$

which, due to the symmetry of logical conjunction, are equivalent. In the case of non minimal parallel compositions it should be intuitive, from every day use, why we wish to enforce the symmetry of this operator. To say for example that an action α is performed in parallel with another β even when the effect of this parallel action is more than the two individual actions cannot really be distinguished from saying that the action β is performed in parallel with α . In order to ensure the validity of this observation we have, via axiom 3, forced the equivalence

$$\alpha \parallel \beta =_{\text{Act}} \beta \parallel \alpha$$

This, if interpreted as identity, which is our intention here, says that both the above compositions refer to the same action.

Non Deterministic choice.

There are many different brands of non determinism ranging from being unsure about when (if at all) various actions occur, through to not caring about which action - usually from a specific collection - is performed. Some formalisms also use it as convenient semantic notion upon which to model concurrency. Since all these different brands of nondeterminism serve different *uses* it will be appropriate to delay discussing and comparing them until we describe the way in which nondeterministic actions are prescribed.

Clearly, depending on what brand of nondeterminism is to be used the structural properties of this combinator will also differ; as far as specifications are concerned we believe that non deterministic actions should represent an - as yet unmade - choice. For example the action of either raising someone's salary or increasing his monthly bonus can, at a certain level of abstraction, be treated as one action if it is known that one of these actions is to be performed but it is not (currently) known which. This composite action may be represented as

Increase-salary + Increase-bonus

which when performed behaves either like Increase-salary or like Increase-bonus, but not both. The benefit of such actions lies in the ability to implement the composite action in numerous ways: we may for example choose to realise it by implementing it as one of the action choices, or by implementing it as an action which behaves in some situations like one action and in other situations like the other. This latter strategy may also be employed as part of the refinement process of specifications which do not want to deal immediately with details pertaining to which action should be performed but do wish to constrain the class of potential actions. We shall see further justification for this view when we look at the normative aspects of such actions.

Like the other combinators the nondeterministic combinator + is a binary function which accepts two actions and returns a third; the axioms that characterise the structural properties of this new action are as follows

$$4.1 \quad ([\alpha] \varphi \wedge [\beta] \psi) \rightarrow ([\alpha+\beta] \varphi \vee \psi)$$

$$5 \quad ([\alpha] \varphi \wedge [\beta] \psi \wedge \neg [\alpha] \psi \wedge \neg [\beta] \varphi) \rightarrow (\neg [\alpha+\beta] \varphi \wedge \psi)$$

$$6 \quad \alpha+\beta =_{\text{Act}} \beta+\alpha$$

Axiom 4.1 ensures that any non deterministic action has the effects of at least one of its constituent actions. Unfortunately, because of the use of disjunction to describe this, this does not rule out the possibility that a non deterministic action can have both the effects of its constituent actions. Leaving things at this would mean that the "+" combinator behaved like a hybrid between non deterministic choice and parallel composition. To some extent, this is redressed in axiom 5; which ensures that so long as neither action in the composition subsumes the effects of the other one, it is not the case that the effects of both actions are realised. Perhaps an example will help to elucidate matters; consider the following action descriptions:

[Strategy-1] P(Take-Bishop)

[Strategy-2] P(Take-Rook)

which form part (describes two strategies) of a chess playing program specification. The effect of employing "Strategy-1" is that it (the program) is permitted to take the opponent's bishop and employing "Strategy-2" leads to a position of play where it is permitted to capture the opponent's rook. A new action of nondeterministically choosing between these strategies - the reader is asked to suspend questions regarding what purpose such an action would serve until we discuss the deontic aspects of non deterministic actions - would according to our axioms have the following description

[Strategy-1 + Strategy-2] P(Take-Bishop) $\vee$ P(Take-Rook)

this, which is a direct result of axiom 4.1, does not prohibit the nondeterministic action of choosing between the two strategies of having the effect of (what could be interpreted as performing both strategies together) producing a scenario in which it is permitted to take the opponent's bishop as well as his rook. However, axiom 5 prohibits this by ensuring that unless the effects of the two actions are the same or one action subsumes the effects of the other, which in our example is not the case, then the nondeterministic action of choosing one of these strategies cannot have both the effects of each strategy.

But what happens in situations where either the effects of the two actions forming the nondeterministic composition are equivalent or one of the action's effects subsumes the others ? Let's analyze both situations. We can depict the situation where a non deterministic composition is formed from two actions such that one action's effects are subsumed by the others as follows:

$$([\alpha] \phi \wedge [\beta] \psi \wedge [\alpha] \psi \wedge \neg [\beta] \phi)$$

Since the effect of the action α also has the same effect as β we can take it that α 's effect

subsumes β 's effect. From axiom 4.1 it should be clear that the nondeterministic action, " $\alpha+\beta$ ", must, when performed, lead to a scenario in which at least (and possibly both) one of ϕ or ψ holds. Therefore, three distinct cases are possible, the effect of performing " $\alpha+\beta$ " is ϕ , or it is ψ or it is both ϕ and ψ . In the case the effect is just ψ it is clear that the action β has been chosen - since we also know that the action β does not produce the effects of α . In the case where the effect of performing " $\alpha+\beta$ " is ϕ it becomes slightly more difficult to say whether only one action is performed. It is of course certain that α is performed but since the effects of α subsume the effects of β how can we be certain that the action " $\alpha+\beta$ " is not the same as the action " $\alpha \parallel \beta$ " ? The simple answer is we can't but we believe it shouldn't matter since the occurrence of β is redundant whether " $\alpha+\beta$ " is constructed as an occurrence of just α or an occurrence of α in parallel with β ; all that matters is that " $\alpha+\beta$ " behaves either like α or like β and this is clearly the case.

A similar line of reasoning can be followed when both the actions involved in a non deterministic composition have equivalent effects. In this case it would not matter if either just one or both the actions involved were performed; the effect would be the same. In other words the action " $\alpha+\beta$ " would be behaviourally indistinguishable from the actions α , β and the minimum parallel composition of these both, " $\alpha \parallel \beta$ ".

In the light of the above analysis it might now be appropriate to reword the intuitive description of the nondeterministic choice combinator. Earlier we said

Performing a non deterministic action amounts to performing one or other (not both) of the actions from which it is formed - usually not knowing which.

which strictly speaking should have been

Performing a non deterministic action amounts to performing an action which is (by virtue of its effect) indistinguishable from one of the actions which constitute the non deterministic composition - usually not knowing which.

Again, this reflects our general view that action combinators are devices to formulate *new* actions whose effects depend (in specific ways) on the effects of the actions named in their formulation. In this sense the combinators are used in a linguistic fashion to name such actions and not as a way of governing the manner in which actions are performed - as for example is the case in programming logics [Gol82].

Like the parallel composition combinator the non deterministic choice combinator is

symmetric and non associative. Symmetry can be shown by considering the following argument: assume the following action descriptions

$$[\alpha] \phi \text{ and } [\beta] \psi$$

then, depending upon which way round we compose and applying axiom 4.1 we get

$$\text{either } [\alpha+\beta] \phi \vee \psi \text{ or } [\beta+\alpha] \psi \vee \phi$$

which are obviously, due to the symmetry of logical disjunction, equivalent. Hence, we may consider these actions to be logically equivalent and axiom 6 to be valid.

The reason for non associativity of the "+" combinator is the same as we gave for the parallel composition combinator. Instead of repeating the argument again we shall just point out that since axiom 4.1 is not an equivalence non minimal compositions can occur and refer the reader to the earlier discussion on the associativity of parallel compositions for an explanation as to why this leads to non associativity of a combinator.

It is however worth saying why axiom 4.1 is not an equivalence. Not unlike parallel compositions the reason is primarily due to the fact that we may have information about the effects of nondeterministic actions over and above the minimal information obtained by axioms four, five and six. For instance there may exist systems in which some actions are only partially nondeterministic; consider for example the following illustration: given two actions a and b, whose sole effects are described as

$$[a] P \text{ and } [b] Q$$

then applying axiom 4.1 would result in the following action description for a + b,

$$[a + b] P \vee Q$$

in addition to this it is quite consistent to also have something of the form

$$R \rightarrow [a + b] Q$$

which fixes the action a + b to have the effect Q in context R. This is quite acceptable if we consider our earlier remarks about refining nondeterministic compositions; using expressions like the above we may fix the interpretation of the action in certain situations and leave, for either implementation or further refinement, how the action behaves in other situations.

But if we were to accept the reverse of 4.1 as valid this would mean that in context R

both the actions a and b are also fixed as having effect Q . Clearly this is only acceptable if we think of the action $a + b$ as an operational choice between a and b , in this case it would be appropriate to regard, in context R , the effects of both a and b as Q . The reader is once again reminded that the novel nondeterministic combinator described here is intended more to highlight our distancing of an operational account of action combinators than describing the conventional (programming) notion of nondeterminism. Following Goldblatt [Gol82] we could characterise the latter as

$$G \quad [\alpha] \varphi \wedge [\beta] \varphi \leftrightarrow [\alpha +_d \beta] \varphi.$$

In addition to this because $+_d$ does not admit compositions other than minimal ones - indeed this is what gives it its operational nature - this combinator would also be associative. The reader should however note that if G were to be incorporated into our system $+_d$ would be slightly different to Goldblatt's version because of our underlying logic of incomplete states. This is easily seen if we consider that in Goldblatt's complete information state based logic we have

$$([\alpha] \varphi \wedge [\beta] \psi) \rightarrow ([\alpha +_d \beta] \varphi \vee \psi) \rightarrow ([\alpha +_d \beta] \varphi \vee [\alpha +_d \beta] \psi)$$

whereas our own stops short of the last consequent and does not in general force a nondeterministic action to decide between two outcomes.

Other examples of this non determinism arise in Dijkstra's notion of guarded commands; the interested reader is referred to [Dij76]. Indeed Goldblatt himself carries out an analysis of how Dijkstra's (IF version of the) guarded command can be defined in terms of $+_d$ (don't care) non determinism.

4.2.2 Normative aspects.

It is reasonable to assume that the introduction of action combinators will effect, in some way or another, the deontic component of our logic. After all it is this part of our formalism that is used to prescribe when actions, including those formulated by the use of action combinators, are permitted or obliged. Surprisingly the combinators impinge very little on the existing formulation of our deontic component; however, as the reader will soon see the deontic properties of non primitive actions raise many important questions which do not arise at the (modal/structural) level of action description. We shall follow the style we adopted in explaining the descriptions of non atomic actions and look at each combinator in turn.

Sequential composition.

As we have seen the general form of a sequentially composed action is $\alpha;\beta$ where α and β are other (perhaps non primitive) actions, because such actions are elements of the syntactic category **Act** they, like any other actions, are subject to the deontic operators P and O . Hence we can represent the property that a sequentially composed action $\alpha;\beta$ is permitted by the formula $P(\alpha;\beta)$ and the property that it is obliged by the formula $O(\alpha;\beta)$. Further due to the axioms of DAL we also have the following:

$$\mathbf{n} \rightarrow (P(\alpha;\beta) \leftrightarrow [\alpha;\beta] \mathbf{n})$$

$$O(\alpha;\beta) \rightarrow P(\alpha;\beta)$$

$$O(\alpha;\beta) \rightarrow \neg \text{Pref}(\alpha;\beta).$$

But, in considering the structural aspects of sequential composition we had to augment the standard modal axioms in order to describe this type of composition in terms of its constituent actions. The next question we must ask ourselves is whether or not the permission (or obligation) of a sequentially composed action is dependent on the permission (or obligation) of its constituent actions and if so how can we represent this dependence. We shall consider the permission operator first.

At the structural level of describing actions we argued that a sequentially composed action $\alpha;\beta$ is equivalent to first doing the action α and then the action β . Taking this into account we may be tempted to say that being permitted to do $\alpha;\beta$ is in fact equivalent to first being permitted to do α and after doing α being permitted to do β , we could represent this symbolically as

$$P1. P(\alpha;\beta) \leftrightarrow P(\alpha) \wedge [\alpha] P(\beta)$$

We think that this would be a mistake; accepting such an equivalence would force us to think of $\alpha;\beta$ as a divisible action consisting of the two separate actions α and β . Earlier we made a vague claim about the essential difference between performing the action $\alpha;\beta$ and the sequence consisting of performing α followed by performing β . This is that $\alpha;\beta$ should be considered as a separate indivisible operation. Assuming that $P(\alpha;\beta)$ holds of a normative scenario (since these are the kind for which permission to do an action α is equivalent to stating that performing α leads to a normative scenario) we have the following equivalence

$$P(\alpha;\beta) \leftrightarrow [\alpha;\beta] \mathbf{n} .$$

Observe that we cannot from this deduce whether or not the action α is permitted; all we know is that we are permitted to do **an** action equivalent to performing α and then performing β . The key point here is that $\alpha;\beta$ as we have defined it is an action (separate to either α or β) which happens to be structurally equivalent to performing β after performing α . Indeed it is quite consistent to have the following

$$P(\alpha;\beta) \wedge \neg P(\alpha)$$

which in any normative scenario may be rewritten as

$$* [\alpha;\beta] \mathbf{n} \wedge [\alpha] \neg \mathbf{n}$$

which in turn, rules out the validity of

$$P1.1 \quad P(\alpha;\beta) \rightarrow P(\alpha) \wedge [\alpha] P(\beta)$$

where P1.1 is the left to right implicand in the equivalence P1 - since this would (whenever we had permission to perform $\alpha;\beta$) force us to accept

$$P(\alpha) \wedge [\alpha] P(\beta)$$

which in a normative scenario is equivalent to

$$[\alpha] \mathbf{n} \wedge [\alpha][\beta] \mathbf{n}$$

and hence inconsistent with *. Hence, if we are to retain the principle that sequentially composed actions are only structurally and not operationally equivalent to their constituent action then it is clear that P1.1 must be dropped.

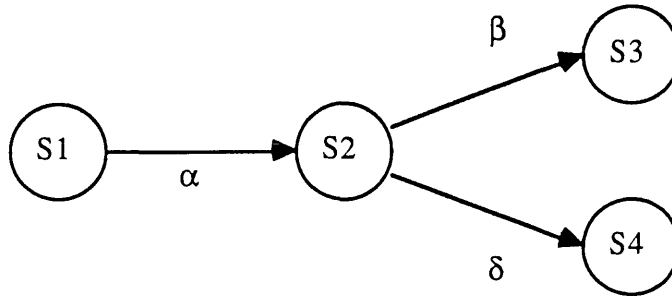
The right to left implicand in P1

$$P1.2 \quad P(\alpha) \wedge [\alpha] P(\beta) \rightarrow P(\alpha;\beta)$$

also interferes with the operational independence of sequentially composed actions. It says that if an action α is permitted and after performing α the action β is permitted then the sequential composition of these actions is also permitted. Again the reason for not wanting this falls on the atomicity of sequentially composed actions. Consider the following illustration, take three actions α , β and δ , with the following properties:

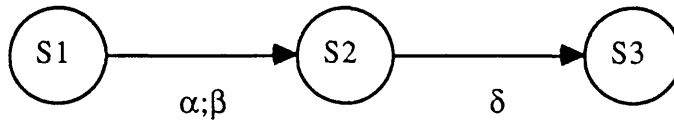
$$P(\alpha) \wedge [\alpha] (P(\beta) \wedge P(\delta))$$

which can be represented in diagram form as:



D1

The arcs in this diagram are labelled by permissible actions and the nodes represent scenarios; compare this with the following diagram:



D2

representing the properties $P(\alpha;\beta)$ and $[\alpha;\beta] P(\delta)$. The difference between permitting sequences of actions and permitting sequentially composed actions should now be easy to see; in D1 although the sequence of performing α and then performing β is permissible so is the sequence of performing α and then performing δ . In D2 being permitted to perform $\alpha;\beta$ does not allow for the possibility of performing the action δ after performing α since there is no separate permission for the action α . It is as though choosing to do a sequentially composed action $\alpha;\beta$ forces us to do something equivalent to performing β after performing α , whereas being allowed to do the sequence α followed by β may provide us with the option of not performing β after performing α .

It seems to us that this shows that permitting a sequentially composed action is a somewhat stronger measure than permitting the corresponding sequence of actions. For this reason, we would not want to adopt P1.2 as an axiom; unfortunately if we look at the structural axiom 1.1 we can see that P1.2 can be logically derived and so must also be considered to be an axiom of our system. The proof of this is as follows:

Proof:

(1)	$(P(\alpha) \wedge [\alpha] P(\beta)) \rightarrow P(\alpha;\beta)$	Hypotheses.
(2)	$P(\alpha) \wedge [\alpha] P(\beta)$	Assumption.
(3)	$[\alpha] \mathbf{n} \wedge [\alpha] P(\beta)$	DAL, (2).
(4)	$[\alpha] (\mathbf{n} \wedge P(\beta))$	SFOAL, (3).
(5)	$[\alpha] (\mathbf{n} \wedge [\beta] \mathbf{n})$	DAL, (4).
(6)	$[\alpha] \mathbf{n} \wedge [\alpha][\beta] \mathbf{n}$	SFOAL, (5).
(7)	$[\alpha][\beta] \mathbf{n}$	$\wedge$ elim, (6)
(8)	$[\alpha;\beta] \mathbf{n}$	Ax. 1.1, (7).
(9)	$P(\alpha;\beta)$	DAL, (8).

The reason for the validity of P1.2 is the unconditional equivalence of axiom 1.1. We can now see why the original formulation of the axiom describing the structural properties of sequentially composed actions was

$$1. \quad P(\alpha;\beta) \rightarrow ([\alpha;\beta] \varphi \leftrightarrow [\alpha][\beta] \varphi)$$

This states that a sequentially composed action is (necessarily) structurally equivalent to performing its constituent actions in sequence, only if it is permitted. This in turn allows us to perform step seven to eight in the above proof only when $P(\alpha;\beta)$ holds. Since

$$[\alpha][\beta] \mathbf{n} \rightarrow [\alpha;\beta] \mathbf{n}$$

is only valid if $P(\alpha;\beta)$ holds.

Suspending for the time being the reason for choosing $P(\alpha;\beta)$ as the condition which restricts (specifically where φ is the logical constant $\mathbf{n}$) the validity of:

$$E1. \quad [\alpha][\beta] \varphi \rightarrow [\alpha;\beta] \varphi$$

we must note that we have so far only justified restricting one side of the equivalence, but axiom 1. also restricts the validity of

$$E2. \quad [\alpha;\beta] \varphi \rightarrow [\alpha][\beta] \varphi$$

The reason for this, again, concerns the operational independence of sequentially composed actions, except this time it has more to do with when a sequentially composed action is not permitted to occur. Consider the situation where we wanted to permit the action sequence of β being performed after α , represented as

$$P(\alpha) \wedge [\alpha] P(\beta)$$

but, wanted to prohibit the the possibility of the sequential composition of these actions, represented as

$$\neg P(\alpha;\beta).$$

Assuming that $\neg P(\alpha;\beta)$ holds of a normative scenario we have the following

$$\neg P(\alpha;\beta) \leftrightarrow [\alpha;\beta] \neg \mathbf{n}$$

further, if we leave E2 unrestricted we also have

$$[\alpha][\beta] \neg \mathbf{n}.$$

On the other hand, assuming that $P(\alpha)$ also holds of a normative scenario we get

$$[\alpha] \mathbf{n} \wedge [\alpha][\beta] \mathbf{n}$$

which is inconsistent with the property that $[\alpha][\beta] \neg \mathbf{n}$. Again the problem lies in the unrestricted validity of E2, and again our answer will be to restrict E2 by the condition $P(\alpha;\beta)$. In the above this means that

$$[\alpha;\beta] \neg \mathbf{n} \text{ does not entail } [\alpha][\beta] \neg \mathbf{n}$$

since $P(\alpha)$ does not hold.

Throughout the above our rationale for the axiom

$$1. \quad P(\alpha;\beta) \rightarrow ([\alpha;\beta] \varphi \leftrightarrow [\alpha][\beta] \varphi)$$

has been the operational independence of sequentially composed actions from the actions from which they are composed. For our present purposes it is sufficient that the reader is convinced that axiom 1. does indeed ensure such independence.

The property that some action is permitted after some other permitted action, can be more appropriately modelled as being permitted to do a sequence of actions much like the way in which we modelled the phenomenon (introduced in the previous chapter) of being obliged to do a sequence of actions. In the binary case we could define this to be

$$PS(< \alpha, \beta >) \leftrightarrow P(\alpha) \wedge [\alpha] P(\beta)$$

where PS (akin to the OS operator) ascribes permission to a sequence of actions.

In general PS is defined as

$$PS(< \alpha >) \leftrightarrow P(\alpha)$$

$$PS(< \sigma, \alpha >) \leftrightarrow \llbracket \sigma \rrbracket P(\alpha)$$

where σ denotes a sequence of actions. The PS operator, like the OS operator, is a convenient shorthand and can be used to prescribe sequences of permitted actions. We shall come across it again, later in this chapter, when we discuss secondary action combinators.

To summarize, we believe that being permitted to do a sequentially composed action should be independent both from the permission of any action mentioned in the composition and the permissibility of the corresponding sequence. The justification for this lies with our view that a sequentially composed action is *an* action which is either permitted or not, and whose performance is atomic. We started out by arguing for the rejection of the equivalence between being permitted to do a sequentially composed action and being permitted to do first one action and then another, this then lead us to restrict the structural properties of the combinator by axiom 1.

Having dealt with the permissibility of sequentially composed actions the obligation of such actions is a relatively straight forward matter. Since $\alpha;\beta$ is an action $O(\alpha;\beta)$ is a valid expression and in accordance with axioms O1 and O2 of the previous chapter we have

$$O(\alpha;\beta) \rightarrow P(\alpha;\beta) \quad \text{and} \quad O(\alpha;\beta) \rightarrow \neg \text{Pref}(\alpha;\beta)$$

these axioms tell us that being obliged to do $\alpha;\beta$ entails being permitted to do $\alpha;\beta$ and not being permitted to do anything else. Hence, any scenario in which there exists an obligation to perform $\alpha;\beta$ there is no permission to perform α and also no permission to perform β . Further, bearing in mind that $\alpha;\beta$ is *an* action, the dynamic aspects of obligations regarding sequentially composed actions are treated in the same manner as primitive actions.

Hence, as in the case of primitive actions, when a sequentially composed action is obliged a transitory suspension of all other permitted actions is imposed until the obligation has been discharged. Further, as a consequence of this, we must also reject any attempt to equate an obligation for a sequentially composed action with the corresponding obligation sequence of actions. The obligation sequence $< \alpha, \beta >$ and an obligation to perform $\alpha;\beta$ can be represented as " $O(\alpha) \wedge [\alpha] O(\beta)$ " and " $O(\alpha;\beta)$ "

In general PS is defined as

$$PS(< \alpha >) \leftrightarrow P(\alpha)$$

$$PS(< \sigma, \alpha >) \leftrightarrow \llbracket \sigma \rrbracket P(\alpha)$$

where σ denotes a sequence of actions. The PS operator, like the OS operator, is a convenient shorthand and can be used to prescribe sequences of permitted actions. We shall come across it again, later in this chapter, when we discuss secondary action combinators.

To summarize, we believe that being permitted to do a sequentially composed action should be independent both from the permission of any action mentioned in the composition and the permissibility of the corresponding sequence. The justification for this lies with our view that a sequentially composed action is *an* action which is either permitted or not, and whose performance is atomic. We started out by arguing for the rejection of the equivalence between being permitted to do a sequentially composed action and being permitted to do first one action and then another, this then lead us to restrict the structural properties of the combinator by axiom 1.

Having dealt with the permissibility of sequentially composed actions the obligation of such actions is a relatively straight forward matter. Since $\alpha;\beta$ is an action $O(\alpha;\beta)$ is a valid expression and in accordance with axioms O1 and O2 of the previous chapter we have

$$O(\alpha;\beta) \rightarrow P(\alpha;\beta) \quad \text{and} \quad O(\alpha;\beta) \rightarrow \neg \text{Pref}(\alpha;\beta)$$

these axioms tell us that being obliged to do $\alpha;\beta$ entails being permitted to do $\alpha;\beta$ and not being permitted to do anything else. Hence, any scenario in which there exists an obligation to perform $\alpha;\beta$ there is no permission to perform α and also no permission to perform β . Further, bearing in mind that $\alpha;\beta$ is *an* action, the dynamic aspects of obligations regarding sequentially composed actions are treated in the same manner as primitive actions.

Hence, as in the case of primitive actions, when a sequentially composed action is obliged a transitory suspension of all other permitted actions is imposed until the obligation has been discharged. Further, as a consequence of this, we must also reject any attempt to equate an obligation for a sequentially composed action with the corresponding obligation sequence of actions. The obligation sequence $< \alpha, \beta >$ and an obligation to perform $\alpha;\beta$ can be represented as " $O(\alpha) \wedge [\alpha] O(\beta)$ " and " $O(\alpha;\beta)$ "

respectively; the incongruence between these two obligation types can be seen by inspecting the actions permitted by them. In the former the obligation to do the action α entails, as a consequence of axioms **O1** and **O2** of DAL, permission to do α but no permission to do any other action (including not permitting the action $\alpha;\beta$) whereas an obligation to do the action $\alpha;\beta$, by the same token, permits the action $\alpha;\beta$ but no other (including the action α). Again the problem arises if we think of an obligation to do a sequentially composed action as a sequence of obligations, the action $\alpha;\beta$ cannot be divide into the actions α and β ; specifically, attempting to impose both the obligations: $O(\alpha)$ and $O(\alpha;\beta)$ leads to inconsistency.

Parallel composition.

Unlike the sequential composition combinator the effects of a parallel action cannot in all cases be reduced to a combination of the effects of its constituent actions. The reason for this lies in the fact that although a parallel action must exhibit the conjunction of the effects of its component actions it may have other effects. This is captured by the following axiom

$$2.1 \quad ([\alpha] \phi \wedge [\beta] \psi) \rightarrow ([\alpha \parallel \beta] \phi \wedge \psi).$$

Unfortunately this also means that if in any normative scenario either α or β are permitted actions then so is the parallel action $\alpha \parallel \beta$. Remembering the DAL axiom

$$\mathbf{n} \rightarrow (P(\alpha) \leftrightarrow [\alpha] \mathbf{n})$$

and assuming that the actions 'a' and 'b' have the following action descriptions

$$[a] P \text{ and } [b] Q$$

then, by axiom 2.1, in any normative scenario if the action 'a' is permitted then so is the action 'a $\parallel$ b'. We can see this by noting that in any normative scenario if 'a' is permitted then we also know that $[a] \mathbf{n}$, if we then apply axiom 2.1 we obtain $[a \parallel b] \mathbf{n}$, which in any normative scenario is equivalent to $P(a \parallel b)$. This would seem to suggest that we adopt the following permission rule for parallel actions

$$P(\alpha) \vee P(\beta) \rightarrow P(\alpha \parallel \beta)$$

We can analyse this by considering two cases: firstly the case where only one but not both of α or β is permitted and secondly the case where both of them are permitted. The first case would validate an axiom of the form

$$P(\alpha) \rightarrow P(\alpha \parallel \beta)$$

Clearly this is not acceptable since this would mean that any permissible action also permitted any parallel composition of that action with any other action. As an illustration of the absurdity of this consider the following action descriptions

$$[\alpha] \varphi \text{ and } [\beta] \neg \varphi$$

Now, from 2.1 we know that the parallel composition of these two actions would lead to inconsistency, but performing either action in isolation does not. Admitting the suggested permission rule would mean that whenever either α or β are permitted so is the inconsistent action of their parallel composition.

The second case in our analysis is where if both actions in a parallel composition are permitted so to is the parallel composition. If this were true it would validate the following permission rule

$$P(\alpha) \wedge P(\beta) \rightarrow P(\alpha \parallel \beta)$$

which says that given permission to perform two separate actions it is also the case that an action equivalent to performing both these actions is also permitted. A simple example should help to show why this is invalid; consider the specification of a robot capable of performing a number of actions but limited by its ability to pick up one item at a time. The above, if valid, would then (wrongly) allow it to pick up a number of items as one indivisible action whenever there existed a choice of items that were permitted to be lifted. We can identify this specific example as an instance of the general case of where given the ability (resources) to perform two actions and also permission to perform both these actions it does not follow that there are the resources to permit the actions to be performed in parallel (either apparent or real).

In order to ensure that parallel actions remain independent of permission to perform the actions from which they are constructed we must modify axiom 2.1 to be

$$2. \quad P(\alpha \parallel \beta) \rightarrow (([\alpha] \varphi \wedge [\beta] \psi) \rightarrow ([\alpha \parallel \beta] \varphi \wedge \psi)).$$

This restricts the validity of 2.1 to those situations in which the parallel action in question is permitted and so prohibits the anomalies we have just seen. Again, we shall defer the rationale behind the precise form of this axiom until we talk about specifications in this extended logic.

Other attempts at trying to establish rules relating the permission of parallel actions to their constituent actions may similarly be shown to be invalid. For example the

following formulation

$$P(\alpha \parallel \beta) \rightarrow (P(\alpha) \wedge P(\beta))$$

is also invalid, consider the following illustration. Assume that as part of our previous robot specification we wish, in certain circumstances, to permit the robot to pick up a screwdriver with one arm and align a pallet with another as one operation, neither of these operations being allowed separately. This can be seen as a specific example of the general case of where some actions are allowed only to be performed in conjunction with others without necessarily being able to be performed individually. Given the relative ease with which such intuitively acceptable arguments can be constructed it would seem inappropriate to accept the proposed axiom as valid. Note that this does not mean that permission to perform parallel actions cannot be characterised in the above (or other) ways for particular actions in specific specifications, simply that such characterisations are not logically valid, that is, no such characterisation holds for all actions in all possible specifications.

Arguments, similar to those above, can also be advanced for the independence of obligations concerning parallel actions. Rather than explore all the possible dependencies between an obliged action and the obligations of its constituent actions, we shall restrict ourselves to showing why the most intuitive is invalid. At first sight it would seem that an obligation to do the parallel action $\alpha \parallel \beta$ meant that both the actions α and β were obliged. But as we know, from axioms O1 and O2 of DAL an obligation to do the action α entails permission to do that action, $P(\alpha)$, and no permission to do any other action (including no permission to do β), whereas an obligation to perform the action β entails permission to do β but no permission to do any other action (including the action α); clearly this leads to inconsistency.

To summarise, any permission or obligation to perform a parallel action should be independent of any permissions or obligations associated with the actions from which it is comprised. It goes without saying that obligatory parallel actions conform to the same dynamic rules as primitive actions.

Non determinism.

Non deterministic actions, like any others, may also be permitted or obliged; in keeping with the other two combinators we are going to argue that the permissibility and obligatoriness of non deterministic actions are independent of the normative properties of

their constituent actions. As usual we shall look at permissions first.

There are many ways we could interpret permission to perform a nondeterministic action; remembering that the purpose of a nondeterministic composition is to represent a choice between performing one of its constituent actions, we may be tempted to think of such permissions as being dependent upon whether or not the actions in the choice are permitted. It has, for instance been suggested, [FH81] and [Wri81] that permission to choose between two actions should be equivalent to having permission to perform both of them. This seems reasonable given English language sentences of the form "Mary is permitted to post the letter or burn it", since this would require, on the face of it, that Mary should have permission to post the letter and permission for her to burn it. Symbolically we would require something of the form

$$P(\text{Mary-Burn-L} \vee \text{Mary-Post-L}) \rightarrow (P(\text{Mary-Burn-L}) \wedge P(\text{Mary-Post-L}))$$

the validity of this as a general condition (and as we shall see later more popularly its Seinsollen counterpart) is an extremely controversial topic in the current deontic logic literature. Perhaps the most enlightening discussion of which can be found in [Jen86]. It seems to us that the controversy surrounding this form of condition stems from regarding a disjunction between actions as an operational choice between performing two actions. Indeed as pointed out in [Jen86] it is sometimes the case that in ordinary language the logical force of the word 'or' is the same as that of 'and'. Mary's situation is a case in point, 'Mary is permitted to do a or q ' is sometimes used to express the same statement as 'Mary is permitted to do a ' and 'Mary is permitted to do b '. In contrast to this our $+$ combinator is used to create actions which internalise this operational choice as part of the action itself. Hence, $P(\alpha+\beta)$ represents permission to do the action $\alpha+\beta$, where performing $\alpha+\beta$ produces at least the effects of performing the action α or the effects of performing β , without necessarily providing an explicit opportunity to choose between performing either. On the other hand something of the form

$$P(\alpha) \wedge P(\beta)$$

represents an explicit choice between two actions. This distinction variously described as local/internal or global/external nondeterminism is also made in CSP/CCS respectively. Although both these formalisms are designed to do quite different jobs from ours some informal analogies can be made. If for example in CSP we wrote a specification involving the nondeterministic composition of two processes represented as $a+b$ then this would force all implementations to always provide for a choice between which

process is executed. This can be compared, in our system, to having permission to perform both a and b , that is $P(a) \wedge P(b)$. In complement to this CSP also provides for nondeterministic compositions of another kind; an expression of the form $\tau.a + \tau.b$, denotes a process which may - by virtue of what is termed a silent τ action - either result in execution of process a or execution of process b . The main difference between this and a process of the form $a+b$ is that an implementation of the former may not preserve the explicit choice present in the latter; it may for example be that the τ action is implemented as always leading to the execution of one of the actions in the choice. Milner refers to this as internal choice since the choice is seen to depend on the silent (internal) τ actions. Considering our earlier remarks about how our nondeterministic combinator internalizes the choice of which action is performed it should not be difficult to see the similarity between being permitted to do a nondeterministic action and the internal nondeterminism of CSP. Similar analogies can be made with the local and global nondeterminism notions of CCS. To summarise we can use the metaphor of Milner's buttons, where a conjunction of permissions such as $P(a) \wedge P(b)$ can be thought of as the existence of two buttons one labelled a and the other b both of which may be depressed whereas $P(a + b)$ denotes the presence of a button labelled $a+b$ which may also be depressed.

Returning now to the burning issue of Mary's postal situation we can see that our nondeterministic combinator is not the intuitive natural language disjunction on actions. So the permission

$$P(\text{Mary-Burn-L} + \text{Mary-Post-L})$$

denotes permission for *an action which results* in Mary either burning the letter or Mary posting it, but it is not necessary from this that Mary has the choice of either burning the letter or burning it. Hence, the classically accepted rule of the form

$$P(\alpha + \beta) \rightarrow (P(\alpha) \wedge P(\beta))$$

is not valid in our system. It should also be clear why the reverse is also unacceptable; again we can employ the metaphor of Milner's buttons - if we are permitted to press a button labelled a and press a button labelled b then it does not follow that we must also be permitted to press a button labelled $a+b$.

In summary, we have seen two kinds of non determinism, the kind which arises naturally as a by-product of concurrency (and more generally by not being in control of *all* of reality) and the kind which helps us to describe not (presently) fully determined

processes. If we are to reflect this difference formally, we will have to ensure - in keeping with the other combinators - the independence of permissions for nondeterministic actions. We can do this by restricting the scope of the structural axiom 4.1 so that we have instead

$$4. \quad P(\alpha+\beta) \rightarrow (([\alpha] \varphi \wedge [\beta] \psi) \rightarrow ([\alpha+\beta] \varphi \vee \psi))$$

The reasons for this restriction are very similar to those we gave in the case of the parallel composition combinator. The primary reason is that it guards against nondeterministic actions inadvertently being permitted when their component actions are; we have already seen why this is not desirable. Note that axiom 5 does not need modification in this way since it does not tell us anything positive about the relationship between the effects of two individual action and their corresponding nondeterministic composition. On the other hand, in any normative scenario if we have the two action descriptions

$$[\alpha] \mathbf{n} \quad \text{and} \quad [\beta] \mathbf{n}$$

then from 4.1 we would have

$$[\alpha+\beta] \mathbf{n} \vee \mathbf{n}$$

which means that (in any normative scenario) permission to perform α and permission to perform β will lead to permission to perform $\alpha+\beta$. We have already seen why this is not desirable; the restriction in axiom 4 prevents this kind of deduction by requiring that a nondeterministic action first be permitted. In the next section we shall see other advantages of this modification.

For reasons similar to those given (and not repeated here) for the other combinators, obligations to perform non deterministic actions are also independent of any logical combination of obligations to do the constituent actions. In addition to this, arguments concerning the static and dynamic aspects of obligations involving non deterministic actions are so similar to those we gave for the parallel composition combinator that we shall not repeat them here.

4.3 Secondary Action Combinators.

The purpose of this section is to discuss a number of operators that ascribe certain normative thresholds as regards the permission or obligation of specific actions. The operators which we are referring to represent the conventional analogues of the P and O operators; instead of permitting or obliging specific actions they are used to describe permissible or obligatory "states of affairs" (properties of scenarios). In [Wri80] Von Wright uses the German term *Tunsollen* to represent action based deontic operators which talk about what is either permitted or obliged to be done, and the term *Seinsollen* for those deontic operators which talk about which "states of affairs" should be permitted or obliged. We have already come across various *Tunsollen* operators (P, O, PS, and OS) but no mention has so far been made of any *Seinsollen* operators; in this section we shall introduce four *Seinsollen* operators one for each of our *Tunsollen* operators. There is however an important difference between our *Seinsollen* operators and those of classical deontic logic (by classical deontic logic we refer of course to the *Seinsollen* system of deontic logic similar to conventional modal logic, see the introduction of the third chapter of this theses for a fuller account); in classical deontic logic no mention is made of specific actions. To say therefore that a certain "state of affairs" is permissible is to state the existence of some anonymous means to realise the scenario (situation) in which that "state of affairs" obtains. In contrast to this our *Seinsollen* operators are defined so that any permitted (obliged) "state of affairs" is translated as the existence of an action (or sequence thereof) which is permitted (obligatory) and when performed results in a scenario of which that "state of affairs" obtains.

The motivation behind having *Seinsollen* operators is that they allow us to dictate the existence of desirable and permissible system scenarios without having to commit ourselves to naming the specific actions that realise those scenarios. This freedom is useful in two ways: firstly it is a more abstract notion of permission than our existing notion of permissible action and secondly it provides us with another view of system specification, by shifting the emphasis away from actions and towards properties of scenarios. This latter point is important if we are to capture certain system properties "naturally" since some system requirements may demand certain properties to be either permitted or obligatory without necessarily stating how such properties are to be brought about.

The remainder of this section is structured as follows: first we shall present the logical apparatus on which our *Seinsollen* operators are founded, we shall then discuss the interpretation of these operators ending with a discussion on how our system compares with others that use deontic *Seinsollen* operators. As usual we shall first present the additional logical language and then the axioms.

Language extension

Syntax.

Logical Operators.

$\text{Per}(_)$, $\text{Obl}(_)$, $\text{Sper}(_)$ and $\text{Sobl}(_)$

Formation rules.

If φ is a CDAL formula then so are

$\text{Per}(\varphi)$, $\text{Obl}(\varphi)$, $\text{Sper}(\varphi)$ and $\text{Sobl}(\varphi)$

Axioms.

- (1) $\text{Per}(\varphi) \leftrightarrow \exists \alpha [\alpha] \varphi \wedge P(\alpha)$
- (2) $\text{Obl}(\varphi) \leftrightarrow \exists \alpha [\alpha] \varphi \wedge O(\alpha)$
- (3) $\text{Sper}(\varphi) \leftrightarrow \exists \sigma \llbracket \sigma \rrbracket \varphi \wedge \text{PS}(\sigma)$
- (4) $\text{Sobl}(\varphi) \leftrightarrow \exists \sigma \llbracket \sigma \rrbracket \varphi \wedge \text{OS}(\sigma)$

Interpretation.

The first thing to note about our *Seinsollen* operators is that they take the form of unary logical connectives, similar, for example, to negation. Like negation they alter the truth of the formula within their scope; the precise manner of this alteration is given by the four definitional axioms above. The rest of this section is concerned with providing an intuitive and technical exploration of these axioms. Our task will be made easier if we do this in two parts dealing first with the operators concerned with permission (Per and Sper) and then with those concerned with obligations (Obl and Sobl).

Per and Sper.

The "Per" operator, short for permissible, is interpreted as guaranteeing the existence of at least one permitted action which when performed results in a scenario with a specific (supplied as argument) property. Looking at axiom (1) we can read an expression of the form $\text{Per}(\phi)$ as follows: there exists an action such that performing that action results in a scenario in which ϕ obtains and further that action is also permitted to occur.

A consequence of this definition is that the "Per" operator talks only about scenarios reachable by performing one atomic action (note that this allows actions formed using primary action combinators); in order to talk about properties of scenarios not attainable by single actions we require the services of the "Sper" operator. "Sper", short for permissible Sequences, is similar to the "Per" operator but ensures the existence of at least one permitted sequence of actions which when performed results in a scenario of which a certain property holds. The reader should be able to see this from axiom (3), where the idea of a permitted sequence is captured by the PS operator (defined earlier when we talked about permission to do sequentially composed actions).

There are a number of interesting theorems that result from these definitions; it is towards the proof of these to which we now turn. The theorems are

$$\mathbf{T1} \quad \text{Per}(\phi \wedge \psi) \rightarrow \text{Per}(\phi) \wedge \text{Per}(\psi)$$

$$\mathbf{T2} \quad \text{Sper}(\phi \wedge \psi) \rightarrow \text{Sper}(\phi) \wedge \text{Sper}(\psi)$$

$$\mathbf{T3} \quad (\text{Per}(\phi) \vee \text{Per}(\psi)) \rightarrow \text{Per}(\phi \vee \psi)$$

$$\mathbf{T4} \quad (\text{Sper}(\phi) \vee \text{Sper}(\psi)) \rightarrow \text{Sper}(\phi \vee \psi)$$

Because of the similarity between these theorems we shall only prove **T1** and **T3** commenting on how similar proofs can be arrived at for theorems **T2** and **T4**. In what follows FOL refers to a conventional first order logic manipulation, similarly SFOAL refers to a manipulation involving the logic of actions presented in chapter two and axioms refer to the definitional axioms of the above *Seinsollen* operators.

Proof of T1.

- | | | |
|----|---|-------------------|
| 1. | $\text{Per}(\phi \wedge \psi) \rightarrow \text{Per}(\phi) \wedge \text{Per}(\psi)$ | <i>Hypothesis</i> |
| 2. | $\neg (\text{Per}(\phi) \wedge \text{Per}(\psi)) \rightarrow \neg \text{Per}(\phi \wedge \psi)$ | 1. and FOL |
| 3. | $\neg (\text{Per}(\phi) \wedge \text{Per}(\psi))$ | <i>Assumption</i> |

- /
- | | | |
|-----|--|----------------------------------|
| 4. | $\neg ((\exists \alpha [\alpha] \varphi \wedge P(\alpha)) \wedge (\exists \beta [\beta] \psi \wedge P(\beta)))$ | 3. <i>and Axiom (1)</i> |
| 5. | $\neg (\exists \alpha [\alpha] \varphi \wedge P(\alpha)) \vee \neg (\exists \beta [\beta] \psi \wedge P(\beta))$ | 4. <i>and FOL</i> |
| 6. | $\forall \alpha (P(\alpha) \rightarrow \neg [\alpha] \varphi) \vee \forall \beta (P(\beta) \rightarrow \neg [\beta] \psi)$ | 5. <i>and FOL</i> |
| 7. | $(P(a) \rightarrow \neg [a] \varphi) \vee (P(a) \rightarrow \neg [a] \psi)$ | 6. <i>and FOL (Univ. Inst.)</i> |
| 8. | $P(a) \rightarrow (\neg [a] \varphi) \vee \neg [a] \psi$ | 7. <i>and FOL</i> |
| 9. | $P(a) \rightarrow \neg ([a] \varphi \wedge [a] \psi)$ | 8. <i>and FOL</i> |
| 10. | $P(a) \rightarrow \neg ([a] \varphi \wedge \psi)$ | 9. <i>and FOAL</i> |
| 11. | $\forall \alpha (P(\alpha) \rightarrow \neg ([\alpha] \varphi \wedge \psi))$ | 10. <i>and FOAL (Univ. Gen.)</i> |
| 12. | $\neg \exists \alpha ([\alpha] (\varphi \wedge \psi) \wedge P(\alpha))$ | 11. <i>and FOL.</i> |
| 13. | $\neg \text{Per}(\varphi \wedge \psi)$ | 12. <i>and Axiom (1).</i> |
| 14. | $\neg (\text{Per}(\varphi) \wedge \text{Per}(\psi)) \rightarrow \neg \text{Per}(\varphi \wedge \psi)$ | 13. <i>discharge assumption.</i> |
| 15. | $\text{Per}(\varphi \wedge \psi) \rightarrow \text{Per}(\varphi) \wedge \text{Per}(\psi)$ | 14. <i>and FOL.</i> |

Q. E. D.

An informal, yet intuitive argument for the validity of **T1** is as follows: the left hand side of the implicand tells us of the existence of an action (which we shall refer to as α) which has two properties, firstly that it is permitted and secondly when performed it results in a scenario in which the property $\varphi \wedge \psi$ holds. Further, we also know that any action α which leads to a scenario in which $\varphi \wedge \psi$ holds also leads to a scenario in which both φ and ψ hold, hence there also exists at least one action (α itself) which has the effect φ and at least one (α) which has the effect ψ . This, together with the fact that the action α is permitted, leads us to the conclusion that both φ and ψ are permissible properties, and this is expressed by the right hand side of **T1**.

The converse of **T1**, represented as

$$\mathbf{T1'} \quad \text{Per}(\varphi) \wedge \text{Per}(\psi) \rightarrow \text{Per}(\varphi \wedge \psi)$$

is not valid; an intuitive counterexample arises when a system admits permission to achieve the property φ and also its negation $\neg \varphi$. This is consistent and may be represented as

$$\text{Per}(\varphi) \wedge \text{Per}(\neg \varphi).$$

If $\mathcal{T}1'$ were valid this would lead to the inconsistent property $\text{Per}(\phi \wedge \neg\phi)$; the argument for the invalidity of $\mathcal{T}1'$ may be further strengthened by noting that (in general) there may exist two separate actions producing distinct consequences without the existence of a third action whose effects subsume the conjunction of these consequences. Consider an example where it is permissible to reach a scenario in which a door is closed and it is also permissible to reach a scenario in which the same door is not closed, via the permitted actions close-door and open-door. It is then nonsense to deduce from this the existence of a permissible scenario in which the door is both not closed and closed,. Accepting $\mathcal{T}1'$ would admit such nonsense.

A proof similar to $\mathcal{T}1$ can be given for $\mathcal{T}2$; the key point to note is that the intuitive argument we have just given works just as well when sequences of permissible actions are involved. The converse of $\mathcal{T}2$ is also invalid for reasons similar to the ones we provided when showing the invalidity of $\mathcal{T}1'$.

For the proof of $\mathcal{T}3$ it will be useful to have the following lemma,

$$\mathcal{L}1 \quad \text{Per}(\phi) \rightarrow \text{Per}(\phi \vee \psi)$$

the proof of $\mathcal{L}1$ is as follows:

- | | | |
|-----|--|-----------------------------------|
| 1. | $\text{Per}(\phi) \rightarrow \text{Per}(\phi \vee \psi)$ | <i>Hypothesis.</i> |
| 2. | $\text{Per}(\phi)$ | <i>Assumption.</i> |
| 3. | $\exists\alpha [\alpha] \phi \wedge P(\alpha)$ | 2. and Axiom 1. |
| 4. | $[a] \phi \wedge P(a)$ | 3. and Exist. Elim. |
| 5. | $P(a)$ | 4. and FOL. |
| 6. | $[a] \phi$ | 4. and FOL. |
| 7. | $\phi \rightarrow \phi \vee \psi$ | <i>Theorem</i> FOL. |
| 8. | $[a] (\phi \rightarrow \phi \vee \psi)$ | 7. and Necessitation SFOAL. |
| 9. | $[a] \phi \rightarrow [a] \phi \vee \psi$ | 8. and $\rightarrow$ Dist. SFOAL. |
| 10. | $[a] \phi \vee \psi$ | 6, 9 and Modus ponens. |
| 11. | $P(a) \wedge [a] \phi \vee \psi$ | 5, 10 and FOL. |
| 12. | $\exists\alpha (P(\alpha) \wedge [\alpha] \phi \vee \psi)$ | 11. and Exist. Intro. |
| 13. | $\text{Per}(\phi \vee \psi)$ | 12. and Axiom 1. |

$$14. \quad \text{Per}(\varphi) \rightarrow \text{Per}(\varphi \vee \psi)$$

2. *Discharge Assumption*

Q.E.D.

Proof of T3.

$$\mathbf{T3} \quad (\text{Per}(\varphi) \vee \text{Per}(\psi)) \rightarrow \text{Per}(\varphi \vee \psi)$$

- | | | |
|----|---|--------------------------------|
| 1. | $(\text{Per}(\varphi) \vee \text{Per}(\psi)) \rightarrow \text{Per}(\varphi \vee \psi)$ | <i>Hypothesis.</i> |
| 2. | $\text{Per}(\varphi) \vee \text{Per}(\psi)$ | <i>Assumption.</i> |
| 3. | $\text{Per}(\varphi)$ | <i>Assumption.</i> |
| 4. | $\text{Per}(\varphi \vee \psi)$ | 3. and L1. |
| 5. | $\text{Per}(\psi)$ | <i>Assumption.</i> |
| 6. | $\text{Per}(\varphi \vee \psi)$ | 5. and L1. |
| 7. | $\text{Per}(\varphi \vee \psi)$ | FOL 2, 4, 6 and $\vee$ Elim. |
| 8. | $(\text{Per}(\varphi) \vee \text{Per}(\psi)) \rightarrow \text{Per}(\varphi \vee \psi)$ | 2. <i>Discharge Assumption</i> |
- Q. E. D.**

The proof of **T3** is simple; it relies mainly on the property - expressed by lemma **L1** - that all permissible actions, and hence permissible properties, preserve the first order theorem of disjunctive weakening. Given this fact it is a relatively straightforward matter to show that the consequent of **T3** may be deduced from each of the disjuncts in the antecedent. Hence, since each disjunct entails the consequent so does the disjunction itself.

However the converse of **T3** is not valid; this is a direct consequence of admitting incomplete information states. To see this we may consider the following counter example: assuming the property $\text{Per}(\varphi \vee \psi)$ we have,

$$\exists \alpha \text{ P}(\alpha) \wedge [\alpha] \varphi \vee \psi.$$

Suppose further that the action α results, when performed, in a scenario from which the property $\varphi \vee \psi$ can be deduced but it cannot be determined in this scenario which of the disjuncts holds. In this situation accepting the validity of

$$\mathbf{T3'} \quad \text{Per}(\varphi \vee \psi) \rightarrow (\text{Per}(\varphi) \vee \text{Per}(\psi))$$

would incorrectly force us to accept either the existence of a permissible action which -

when performed - leads to a scenario in which ϕ holds or the existence of a permissible action which leads to a scenario in which ψ holds. In other words accepting $\mathcal{T3}'$ would force us into systems with complete information scenarios. The reader is referred to the invalidity of the following expression

$$[\alpha] \phi \vee \psi \rightarrow ([\alpha] \phi \vee [\alpha] \psi)$$

which is discussed more fully in the second chapter of this work.

The distribution of permissible properties over disjunction is extended in $\mathcal{T4}$ to deal with properties which may require a sequence of actions to be realised. We shall not give the proof of $\mathcal{T4}$ since it is very similar to that of $\mathcal{T3}$, the extra complication of permissible sequences of actions does not alter the above intuitive arguments. Note also that the converse of $\mathcal{T4}$ is invalid for the same reasons we gave in support of the invalidity of $\mathcal{T3}'$.

Obl and Sobl.

The interpretation usually given to obligatory "states of affairs" is (in classical deontic logic) similar to the modal concept of necessity; in this alethic setting something which is obligatory is translated as being true (or being provable) of all realisable scenarios. No mention is made of what is meant by realisable scenarios. Our definition of an obligatory state of affairs is captured by the Obl operator which corrects this situation by basing itself on the idea of an obligatory action. As we can see by axiom 2 the notion of an obligatory property means that there exists an action which is obligatory - and because of the static aspects of obligatory actions there can only be one such action - and results (when performed) in the stated property.

At first this definition may seem far from ideal and even counter-intuitive; consider an example where it is obligatory that a department store sells all of its last years stock; from here on we refer to this property as "sell-out". There may be a number of permitted ways in which this may be done, for example by reducing prices or holding back new stock until last years had been depleted, and it would seem intuitive to allow any one of these methods to be employed. But our definition states that being obliged to "sell-out" is equivalent to the existence of an action which is obliged and results in the desired property of "sell-out".

In defence of our definition we would remind the reader of our view that obligatory actions are inherently transitory phenomena, and assert that we also maintain this view when it comes to obligatory properties. In other words we regard an obligatory property as a property that (temporarily) takes priority over all other properties and must be realised at the next opportunity, after which things may return to normal. As a result of this position we have built the notion of an obligatory property on top of the notion of an obligatory action, the precise construction of which is given by axiom 2.

Note that we allow for the fact that many actions may achieve an obligatory property but at the same time require that at least and (to retain consistency) at most one of them is obligatory. A specification may be more specific and choose which action in particular is obliged, or it may not, just so long as one and only one of the actions, which can realise the obligatory property, is obliged.

The 'Sobl' operator is a generalised version of the 'Obl' operator; 'Sobl' extends 'Obl' in much the same way as 'Sper' extends the 'Per' operator, that is, it allows us to assert obligatory properties that may not be realisable in just one atomic action but may instead require a sequence of actions. The definition of 'Sobl', provided by axiom (4), is similar to that of 'Obl' differing only in that it ensures the existence of an obligatory sequence (see the definition of the 'OS' operator) of actions which when performed (in sequence) realise the desired obligatory property rather than a single atomic action doing so. The following theorems result as a consequence of our definitions of 'Obl' and 'Sobl' given by axioms (2) and (4) respectively.

$$\mathbf{T5} \quad \text{Obl}(\phi \wedge \psi) \leftrightarrow \text{Obl}(\phi) \wedge \text{Obl}(\psi)$$

$$\mathbf{T6} \quad \text{Sobl}(\phi \wedge \psi) \rightarrow \text{Sobl}(\phi) \wedge \text{Sobl}(\psi)$$

$$\mathbf{T7} \quad (\text{Obl}(\phi) \vee \text{Obl}(\psi)) \rightarrow \text{Obl}(\phi \vee \psi)$$

$$\mathbf{T8} \quad (\text{Sobl}(\phi) \vee \text{Sobl}(\psi)) \rightarrow \text{Sobl}(\phi \vee \psi)$$

Proof of

$$\mathbf{T5} \quad \text{Obl}(\phi \wedge \psi) \leftrightarrow \text{Obl}(\phi) \wedge \text{Obl}(\psi)$$

A proof of **T5** may be given in two parts, dealing with each implicand of the equivalence in turn.

Proof of

$$\text{Obl}(\varphi \wedge \psi) \rightarrow \text{Obl}(\varphi) \wedge \text{Obl}(\psi)$$

- | | | |
|-----|--|---------------------------|
| 1. | $\text{Obl}(\varphi \wedge \psi) \rightarrow \text{Obl}(\varphi) \wedge \text{Obl}(\psi)$ | <i>Hypothesis</i> |
| 2. | $\neg (\text{Obl}(\varphi) \wedge \text{Obl}(\psi)) \rightarrow \neg \text{Obl}(\varphi \wedge \psi)$ | 1. and FOL |
| 3. | $\neg (\text{Obl}(\varphi) \wedge \text{Obl}(\psi))$ | <i>Assumption</i> |
| 4. | $\neg ((\exists \alpha [\alpha] \varphi \wedge \text{O}(\alpha)) \wedge (\exists \beta [\beta] \psi \wedge \text{O}(\beta)))$ | 3. and Axiom (2) |
| 5. | $\neg (\exists \alpha [\alpha] \varphi \wedge \text{O}(\alpha)) \vee \neg (\exists \beta [\beta] \psi \wedge \text{O}(\beta))$ | 4. and FOL |
| 6. | $\forall \alpha (\text{O}(\alpha) \rightarrow \neg [\alpha] \varphi) \vee \forall \beta (\text{O}(\beta) \rightarrow \neg [\beta] \psi)$ | 5. and FOL |
| 7. | $(\text{O}(a) \rightarrow \neg [a] \varphi) \vee (\text{O}(a) \rightarrow \neg [a] \psi)$ | 6. and FOL (Univ. Inst.) |
| 8. | $\text{O}(a) \rightarrow (\neg [a] \varphi) \vee \neg [a] \psi$ | 7. and FOL |
| 9. | $\text{O}(a) \rightarrow \neg ([a] \varphi \wedge [a] \psi)$ | 8. and FOL |
| 10. | $\text{O}(a) \rightarrow \neg ([a] \varphi \wedge \psi)$ | 9. and FOAL |
| 11. | $\forall \alpha (\text{O}(\alpha) \rightarrow \neg ([\alpha] \varphi \wedge \psi))$ | 10. and FOAL (Univ. Gen.) |
| 12. | $\neg \exists \alpha [\alpha] (\varphi \wedge \psi) \wedge \text{O}(\alpha)$ | 11. and FOL. |
| 13. | $\neg \text{Obl}(\varphi \wedge \psi)$ | 12. and Axiom (2). |
| 14. | $\neg (\text{Obl}(\varphi) \wedge \text{Obl}(\psi)) \rightarrow \neg \text{Obl}(\varphi \wedge \psi)$ | 13. discharge assumption. |
| 15. | $\text{Obl}(\varphi \wedge \psi) \rightarrow \text{Obl}(\varphi) \wedge \text{Per}(\psi)$ | 14. and FOL. |
- Q. E. D.

Proof of

$$\text{Obl}(\varphi) \wedge \text{Obl}(\psi) \rightarrow \text{Obl}(\varphi \wedge \psi)$$

The proof of the above implicand relies on a static property of obligations, namely that only one action may be obliged in any particular scenario. The following lemma will be helpful during the proof:

$$\mathbf{L2.} \quad \forall \alpha, \beta (\text{O}(\alpha) \wedge \text{O}(\beta)) \rightarrow \alpha = \beta.$$

in the proof below we shall follow the classical *reductio ad absurdum* strategy - this relies on proving that the negation of our original hypothesis leads to inconsistency and then invoking the law of excluded middle to assert the validity of the original hypothesis, namely **L2**.

1.	$\forall \alpha, \beta \neg (O(\alpha) \wedge O(\beta) \rightarrow (\alpha = \beta))$	<i>Hypothesis</i>
2.	$\neg (O(a) \wedge O(b) \rightarrow (a = b))$	1. and Univ. Inst.
3.	$O(a) \wedge O(b) \wedge \neg (a = b)$	2. and FOL.
4.	$O(a)$	3. and FOL.
5.	$O(b)$	3. and FOL.
6.	$\neg (a = b)$	3. and FOL.
7.	$\forall \alpha (O(\alpha) \rightarrow P(\alpha))$	O1 DAL.
8.	$\forall \alpha (O(\alpha) \rightarrow \neg \text{Pref}(\alpha))$	O2 DAL.
9.	$O(a) \rightarrow P(a)$	7. Univ Inst.
10.	$O(a) \rightarrow \neg \text{Pref}(a)$	8. Univ Inst.
11.	$P(a)$	4,9 Modus ponens.
12.	$\neg \text{Pref}(a)$	4,10 Modus ponens.
13.	$O(b) \rightarrow P(b)$	7. Univ Inst.
14.	$P(b)$	5,13 Modus ponens.
15.	$\neg \text{Pref}(a) \wedge P(b) \wedge \neg (a = b)$	6,12,14. FOL.
16.	$\neg (\exists \beta (P(\beta) \wedge \neg (a = \beta))) \wedge P(b) \wedge \neg (a = b)$	15. Def. Pref - DAL.
		Q.E.D.

We are now in a position to proceed with the proof proper:

1.	$\text{Obl}(\varphi) \wedge \text{Obl}(\psi)$	<i>Hypothesis</i>
2.	$(\exists \alpha [\alpha] \varphi \wedge O(\alpha)) \wedge (\exists \beta [\beta] \psi \wedge O(\beta))$	1 and Axiom 2.
3.	$(\exists \alpha O(\alpha)) \wedge (\exists \beta O(\beta))$	2 and SFOAL.
4.	$\alpha = \beta$	3 and L2.
6.	$(\exists \alpha [\alpha] \varphi \wedge O(\alpha)) \wedge (\exists \alpha [\alpha] \psi \wedge O(\alpha))$	2 and 4.
7.	$(\exists \alpha ([\alpha] \varphi \wedge \psi) \wedge O(\alpha))$	6 and SFOAL.
8.	$\text{Obl}(\varphi \wedge \psi)$	7 and Axiom 2.
		Q.E.D.

The above proves that obligatory conjunctive properties can be equivalently expressed as

a conjunction of obligatory properties. We can see this more easily if, like in the above proof, we inspect each side of the equivalence in turn. Following the order of the proof we shall first examine

$$\mathbf{T5.1} \quad \text{Obl}(\phi \wedge \psi) \rightarrow \text{Obl}(\phi) \wedge \text{Obl}(\psi).$$

This says that if there is an obligation to realise a scenario with property $\phi \wedge \psi$ then both ϕ and ψ are obligatory properties. If we assume the antecedent of **T5.1** we are assuming the existence of an action which is obligatory and whose effect is $\phi \wedge \psi$. On the other hand the consequent of **T5.1** says that there exists an (obligatory) action whose effect is ϕ and an obligatory action whose effect is ψ . But, because of the properties of actions, we also know that any action which leads to a scenario in which $\phi \wedge \psi$ holds also results in a scenario in which ϕ holds and also leads to a scenario in which ψ holds. Hence given any action which satisfies the existence criteria given in the antecedent of **T5.1** *that* action also satisfies both the existence criteria of the consequent. In fact the intuitive justification of this implicand is very similar to that which we gave in the proof of **T1**. Note that this argument does not hinge on the fact that only one action may be obliged at any one time (although this is in fact the case).

We come now to the second part of the equivalence which involves proving

$$\mathbf{T5.2} \quad \text{Obl}(\phi) \wedge \text{Per}(\psi) \rightarrow \text{Obl}(\phi \wedge \psi)$$

This says that whenever it is the case that both ϕ and ψ are obligatory properties it follows that $\phi \wedge \psi$ is an obligatory property. Here the antecedent asserts the existence of two actions (both obligatory) one resulting in a scenario in which ϕ holds and the other resulting in a scenario in which ψ holds. The crucial point here is that these actions must be identical (proved in the lemma given earlier) since only one action can be obligatory at any given time. In other words there must be one and only one obligatory action, such that when it is performed it leads to a scenario in which both ϕ and ψ hold. This is exactly what the consequent of the implicand captures.

The reader should note that the corresponding permission expression

$$\mathbf{T1'} \quad \text{Per}(\phi) \wedge \text{Per}(\psi) \rightarrow \text{Per}(\phi \wedge \psi)$$

is not valid because more than one permissible action may exist at any one time meaning that two distinct actions could satisfy the antecedent neither of which need also satisfy the consequent of the above implicand.

Rather than labouring through the proofs of the remaining theorems we shall restrict ourselves to commenting on only the important points.

The proof of **T6** is similar to the proof of **T5.1** and relies on the same intuitive arguments. However the converse of **T6**, unlike the converse of **T5.1**, is not valid essentially due to the fact that the lemma **L1** does not extend to obligation sequences. It is for instance quite reasonable for an obligation sequence to lead to a property ϕ and a subsequence of the same obligation sequence to lead to a scenario in which ψ holds without there existing an obligation sequence in which both $\phi \wedge \psi$ holds.

The two remaining theorems **T7** and **T8** follow the same proof outlines as theorems **T3** and **T4**, for much the same intuitive reasons. Indeed a substitution of operators: Obl for Per and O for P in the proof of **T3** gives a proof of **T7** and a substitution of the operators: Sobl for Sper and OS for PS in the proof of **T4** results in a proof of **T8**. The converse of both **T7** and **T8** are invalid for the same reasons as were the converses of **T3** and **T4**, and the counter examples that we gave for the latter apply equally well to the former.

In the next section we shall see how the machinery of primary and secondary combinators helps us to write more expressive specifications.

4.4 CDAL (*Combinatory Deontic Action Logic*).

In the previous two sections we extended DAL in two dimensions: first to cope with the notion of more complex actions, expressed as specific combinations of other actions and then to incorporate some means for talking about the permission/obligation of general properties. The purpose of this section is twofold, firstly to merge the extensions of the last two sections into *an* extension of DAL - called CDAL (Combinatory DAL) and secondly to see how this extended formalism can be used to write more expressive specifications. Again rather than rewrite the DAL component of CDAL we shall focus mainly on the extension.

Language extension.

DAL: All the syntax and formation rules of DAL,

plus the following logical functions:

Functions:

; of type $\langle \text{Act}, \text{Act}, \text{Act} \rangle$

|| of type $\langle \text{Act}, \text{Act}, \text{Act} \rangle$

+ of type $\langle \text{Act}, \text{Act}, \text{Act} \rangle$

Logical Operators.

Per(_), Obl(_), Sper(_) and Sobl(_)

PS(_)

Formation rules.

If σ is an action sequence then

PS(σ) is a CDAL formula.

If φ is a CDAL formula then so are

Per(φ), Obl(φ), Sper(φ) and Sobl(φ).

Axioms.

DAL: All the axioms of DAL *plus* the ones below:

- (1) $P(\alpha;\beta) \rightarrow ([\alpha;\beta] \varphi \leftrightarrow [\alpha][\beta] \varphi)$
- (2) $P(\alpha \parallel \beta) \rightarrow ([\alpha] \varphi \wedge [\beta] \psi \rightarrow ([\alpha \parallel \beta] \varphi \wedge \psi))$
- (3) $\alpha \parallel \beta =_{\mathbf{Act}} \alpha \parallel \beta$
- (4) $P(\alpha+\beta) \rightarrow ([\alpha] \varphi \wedge [\beta] \psi \rightarrow ([\alpha+\beta] \varphi \vee \psi))$
- (5) $([\alpha] \varphi \wedge [\beta] \psi \wedge \neg [\alpha] \psi \wedge \neg [\beta] \varphi) \rightarrow (\neg [\alpha+\beta] \varphi \wedge \psi)$
- (6) $\alpha+\beta =_{\mathbf{Act}} \alpha+\beta$
- (7) $PS(<\alpha>) \leftrightarrow P(\alpha)$
- (8) $PS(<\sigma, \alpha>) \leftrightarrow \llbracket \sigma \rrbracket P(\alpha)$
- (9) $Per(\varphi) \leftrightarrow \exists \alpha [\alpha] \varphi \wedge P(\alpha)$
- (10) $Obl(\varphi) \leftrightarrow \exists \alpha [\alpha] \varphi \wedge O(\alpha)$
- (11) $Sper(\varphi) \leftrightarrow \exists \sigma \llbracket \sigma \rrbracket \varphi \wedge PS(\sigma)$
- (12) $Sobl(\varphi) \leftrightarrow \exists \sigma \llbracket \sigma \rrbracket \varphi \wedge OS(\sigma)$

The above language and axioms should by now be familiar: axioms one to six deal with the description of primary action combinators, seven and eight define the permissible sequence operator PS (similar to the Obligation Sequence operator OS) and finally axioms nine to twelve define the *Siensollen* operators.

The definition of model for a CDAL specification - a specification written in CDAL - remains unchanged from that of a DAL specification. There is the usual definition of model structure (a collection of scenarios and functions) with all the previous model constraints, except now since we have increased the action category by including non primitive actions we also have functions for these more complex actions as well. Further because non primitive actions are operationally atomic they are subject to the same constraints as primitive ones; in addition to these common constraints complex actions are further constrained by axioms one to six above. We have already discussed in some detail the motivation for, and the technical aspects of, these axioms. We shall now consider their role from a specification prespective.

In addition to the basic logical apparatus every purposeful CDAL specification will also include some extralogical language with which to refer to the domain under consideration. As we saw earlier an important part of this extralogical language deals with the naming of actions; primary action combinators extend our ability to name actions in a very powerful way. There are two views we can take of such combinators: from a logical viewpoint they provide a *potential* extension of the primitive action category so as to include non primitive actions, whilst from a specification viewpoint they can be seen as a dynamic means of generating names for interesting combinations of other actions. Both these views are mutually consistent and valid.

The key point here is that a specification is concerned only with those non primitive actions which it explicitly names as part of its extralogical language and not all the potential combinations of its initial primitive set. To illustrate this consider the following, less restrictive version, of axiom two above,

$$2.1 \quad ([\alpha] \varphi \wedge [\beta] \psi) \rightarrow ([\alpha \parallel \beta] \varphi \wedge \psi).$$

This says that for any two actions whose effects are φ and ψ respectively then the action which is the parallel composition of these two has the effect of producing a scenario where $\varphi \wedge \psi$ holds. Note that this does not mean that such a parallel action exists (is part of the system we are trying to specify) just that if it did its effect would be to produce a certain result. This is quite a life saver since a good many combinations of actions are either nonsensical or produce inconsistent effects.

In any specific specification, we would only be able to make use of an axiom such as 2.1 if together with two action names, a and b say, we also had the action name $a \parallel b$ as part of the extralogical language of the specification. This means that many spurious combinations of actions are ruled out at the linguistic level, but this only provides a partial solution to dealing with unwanted combinations of actions. Assuming a particular parallel action is included in the language of a specification then according to 2.1 its minimal composite effect will vary in accordance with the effects of the actions from which it is composed. But how can we be sure that the composite effect will be sensible in all situations ? Ultimately this task will rest with the specifier; we can however aid him by restricting the scope of 2.1 to only those parallel actions which are permitted to be performed. Ideally we should have some notion of when actions may be combined with others but we felt that for our present purposes it was sufficient to express this as being permitted (to be performed) on the grounds that it would be pointless to compose actions without the chance of them being also being performed. Apart from the reasons we

discussed earlier this is why axiom 2 is the following restricted form of 2.1

$$(2) \quad P(\alpha \parallel \beta) \rightarrow (([\alpha] \phi \wedge [\beta] \psi) \rightarrow ([\alpha \parallel \beta] \phi \wedge \psi)).$$

Such a restriction provides a specifier with more control over when parallel compositions of actions have a certain minimal effect; in other words it allows him/her to circumvent the logically necessary property that the effects of parallel actions are always (at least) a conjunction of the effects of their constituent actions. Note that this does not mean that parallel actions are only allowed to have this property when they are permitted, simply that if they are permitted then this property does hold. The following example should illustrate this more concretely: consider the following specification fragment in which A and B are actions and P, Q, R and S are some abstract properties,

$$S \rightarrow [A] P \quad [B] Q$$

$$R \rightarrow [B] \neg P$$

where the action $A \parallel B$ is included in the language of our specification but has no explicit description. Then, according to 2.1 we would have the following logically necessary description for $A \parallel B$,

$$S \rightarrow [A \parallel B] (P \wedge Q) \quad \text{and} \quad R \rightarrow [A \parallel B] (P \wedge Q \wedge \neg P)$$

clearly the effect of $A \parallel B$ when property R holds leads to inconsistency and hence an inconsistent specification. In contrast to this consider the slightly different specification below

$$S \rightarrow [A] P \quad [B] Q$$

$$R \rightarrow [B] \neg P \quad S \rightarrow \text{Per}(A \parallel B) \quad \text{and} \quad R \rightarrow \neg \text{Per}(A \parallel B)$$

in conjunction with axiom 2. Not in possession of any other information about the parallel action we will only be able to conclude

$$S \rightarrow [A \parallel B] (P \wedge Q)$$

and not the undesirable inconsistency. Similar arguments can be constructed for the other combinators; it may for example be that a sequential composition cannot be implemented and a sequence of actions is the only thing available. In such a situation it would be incorrect to force some description of a sequential composition if that composition cannot take place. We hope, that from the above intuition and the more technical arguments of the earlier sections the reader can see the reasons for choosing the above axioms.

4.5 An Example.

In order to illustrate the usefulness of this extension we have chosen to do a partial specification (entitled Netspec) of an electronic mailing network. There are a number of reasons for our choice: it is a familiar computing science domain with a large amount of research interest; it provides a rich domain of non primitive actions - the conventional device is a transaction (a package of primitive actions), and finally, it allows us to focus on the use of combinators without the need for much peripheral description. The network is to have a centralised routing/management system which decides when, and to where, messages are moved, so that they may reach their destinations. In the specification below we shall restrict ourselves (wherever possible) to characterising the valid routes a message may/must travel in this network, thus prescribing how any correct implementation of the network must operate. We proceed in the usual manner presenting first the language and then the axioms.

Netspec.

Language.

Sorts: Message, Priority, Node and Act.

Constants for sort:

Message: None.

Priority: NM, IM, UG.

Node: A, B, C, D, E, F, G.

Function names:

Org: of type < Message, Node >.

Des: of type < Message, Node >.

Id: of type < Message, Integer >.

Predicate names:

At: of type < Message, Node >.

Prt: of type < Message, Priority >.

Action names:

Send: of type $\langle \text{Message}, \text{Node}, \text{Node} \rangle$.

Disp: of type $\langle \text{Message}, \text{Node} \rangle$.

Enter: of type $\langle \text{Message}, \text{Node} \rangle$.

Axioms.

1. $\forall m \forall n, n' \quad (\text{At}(m, n) \wedge \text{At}(m, n')) \rightarrow (n = n')$
2. $\forall m, m' \quad \neg (m = m') \rightarrow \neg (\text{Id}(m) = \text{Id}(m'))$.
3. $\forall m \forall n \quad \text{Des}(m) = n \rightarrow \neg P(\text{Enter}(m, n))$.
4. $\forall m \forall n, n' \quad n = n' \rightarrow \neg P(\text{Send}(m, n, n'))$.
5. $\forall m \forall n, n' \quad \neg (\text{At}(m, n)) \rightarrow \neg P(\text{Send}(m, n, n'))$.
6. $\forall m \quad (\text{Prty}(m, \text{UG}) \rightarrow \text{Prty}(m, \text{IM})) \wedge (\text{Prty}(m, \text{IM}) \rightarrow \text{Prty}(m, \text{NM}))$.
7. $\forall m \forall n, n' \quad [\text{Enter}(m, n)] \text{At}(m, n) \wedge \text{Org}(m) = n \wedge \neg P(\text{Enter}(m, n'))$
8. $\forall m \forall n, n' \quad [\text{Disp}(m, n)] P(\text{Enter}(m, n'))$
9. $\forall m \forall n, n' \quad [\text{Send}(m, n, n')] \neg \text{Sper}(\text{At}(m, n))$.
10. $\forall m \forall n, n' \quad (\neg \text{Des}(m, n') \rightarrow [\text{Send}(m, n, n')] (\neg \text{At}(m, n) \wedge \text{At}(m, n'))$.
11. $\forall m \forall n, n', n'' \quad \text{Des}(m, n') \rightarrow$
 $([\text{Send}(m, n, n')] \neg (\text{At}(m, n'') \wedge \text{Sper}(\text{O}(\text{Disp}(m, n'))))$.
12. $\forall m \forall n, n' \quad \text{Prty}(m, \text{NM}) \rightarrow ($
 $(\text{At}(m, A) \rightarrow P(\text{Send}(m, A, B))) \wedge (\text{At}(m, B) \rightarrow P(\text{Send}(m, B, A))) \wedge$
 $(\text{At}(m, A) \rightarrow P(\text{Send}(m, A, C))) \wedge (\text{At}(m, C) \rightarrow P(\text{Send}(m, C, A))) \wedge$
 $(\text{At}(m, C) \rightarrow P(\text{Send}(m, C, D))) \wedge (\text{At}(m, D) \rightarrow P(\text{Send}(m, D, C))) \wedge$
 $(\text{At}(m, C) \rightarrow P(\text{Send}(m, C, E))) \wedge (\text{At}(m, E) \rightarrow P(\text{Send}(m, E, F))) \wedge$
 $(\text{At}(m, F) \rightarrow P(\text{Send}(m, F, D))) \wedge (\text{At}(m, F) \rightarrow P(\text{Send}(m, F, G))) \wedge$
 $(\text{At}(m, G) \rightarrow P(\text{Send}(m, G, F))) \wedge (\text{At}(m, C) \rightarrow \neg P(\text{Send}(m, C, B)))$.
13. $\forall m \forall n, n' \quad (\text{At}(m, E) \wedge \text{Prty}(m, \text{IM})) \rightarrow P((\text{Send}(m, E, D)))$.

14. $\forall m \forall n, n' (At(m, C) \wedge Prty(m, IM)) \rightarrow$
 $P((Send(m, C, E) ; Send(m, E, F)) + (Send(m, C, F))).$
15. $\forall m Prty(m, IM) \rightarrow [Enter(m, A)] P((Send(m, A, D)) + (Send(m, A, E))).$
16. $\forall m Prty(m, IM) \rightarrow [Enter(m, G)] Per(At(m, D) \vee At(m, E)).$
17. $\forall m \forall n Prty(m, UG) \rightarrow Obl(\neg At(m, n)).$
18. $\forall m \forall n, n', n'' (Prty(m, UG) \wedge At(m, n) \wedge PS(< Send(m, n, n'), Send(m, n', n'') >)$
 $\rightarrow P(Send(m, n, n') ; Send(m, n', n'')).$
19. $\forall m, m' \forall n, n', n'', n'''$
 $(Prty(m, UG) \wedge Prty(m', UG) \wedge At(m, n) \wedge At(m', n') \wedge \neg (m = m')$
 $\wedge P(Send(m, n, n'') \wedge P(Send(m, n', n''')))$
 $\rightarrow P(Send(m, n, n'') \parallel Send(m, n', n''')).$

Perhaps the most instructive way we can begin our discussion of *Netspec* is by presenting an informal account of what the above specification is intended to capture. This can then be followed by a description of the concrete syntax we have used, ending with a discussion of the specification axioms proper.

Broadly speaking, the purpose of the network is to provide a service for transferring messages between a number of nodes. In most networks this is a distributed activity requiring synchronisation and other protocol mechanisms; since we will not be concerned with such matters here we shall leave it open as to whether or not the routing is done centrally. A message is entered into the network at a particular node and is then sent by some route - some intermediary nodes - to its destination. In order to deliver messages optimally most networks take into account the current position of the message and the prescribed routes available to it (from that position). The purpose of the above specification is to prescribe the possible routes a message can take and for us this will depend upon a number of factors such as the current position of the message, its destination, its origin and its priority. Note that we are not in the business of specifying how the network delivers messages - since this would require a specific scheduling strategy. When a message is delivered to its destination node it is displayed and is considered to have then left the network. There is no notion of actively receiving a message.

The language of the specification consists of three application dependent sorts; messages, nodes and priorities, with their obvious interpretations. Each message has associated with it an origin node, a destination node and a priority. Since messages have only one destination and one origin this information is modelled by two functions, *Org* and *Des*, both of which, given a message, return either the node where the message entered the network (*Org*) or the node where it must be delivered (*Des*).

There are three elements of the sort priority: NM, short for normal priority, IM when messages are important and UG when messages are considered to be urgent. Since a message may hold more than one priority, for reasons we shall see later, the priority of a message is modelled by the two place predicate '*Prty(m,p)*' which holds when the message '*m*' has priority '*p*'.

Apart from this "user" oriented information about messages the network must also possess information about the current position(s) of the messages still waiting to be delivered. This information is modelled by the predicate '*At(m,n)*' which holds whenever a message '*m*' is at a node '*n*'. Although any message can only be at (at most) one node we do not use a function for this purpose because there are times when we do not want a message to be at any node - although we will require one more axiom to ensure that all messages are, at most, only present at one node, at any given time.

There are three primitive actions in the action category *Act*, these are: Enter, Send and Display. The Enter action is used to model the initial act of injecting a message into the network. It has two parameters, a message '*m*' and a node '*n*' such that *Enter(m, n)* is read "message *m* is entered into the network at node *n*". The Send action represents the act of sending a message between two nodes it has three parameters a message and two nodes such that *Send(m, n, n')* is interpreted as " the message *m* is sent from node *n* to node *n'*". Lastly the Display action represents the action of a message leaving the network: it has two parameters, a message and a node, the idea being that whenever a message *m* is displayed at a node *n* it is considered to have been received at its destination and need no longer be considered by the network. Apart from these primitive actions Netspec also incorporates certain combinations of these primitive ones but descriptions of these are best left until we encounter them in the body of the specification.

Before we look at the individual axioms of Netspec it will be worth pointing out the general structure of the specification: the first six axioms are of a general nature, axioms

seven to eleven describe the three primitive actions, axiom twelve expresses the routes available to a normal message (of priority NM), the next four axioms up to and including axiom sixteen prescribe the routes available to important messages (of priority IM) and finally the last three axioms deal with routes for urgent messages (of priority UG). We start by looking at the general axioms.

General axioms.

Axiom one is a straight forward requirement on messages - any message m may be at, at most one node.

Axiom two states that no two messages are ever the same, in other words each message is unique, this is common condition in electronic networks. The purpose of this is to ensure that messages of otherwise identical content can be distinguished.

Axiom three says that any message which has destination node n cannot be entered into the network at node n . The redundancy of such an operation should be evident, especially when we consider that the purpose of the network is to deliver messages to their destination nodes - something vacuously satisfied if the message enters the network at its destination node!

Axiom four is a generalisation of axiom two and prevents the network from sending (in one step) messages between the same node; in other words a message is prohibited from travelling to the node from which it is being sent. This seems like a sensible restriction since it rules out many of the potentially redundant routes that a message can travel, including infinite loops. Later we shall see that this axiom in conjunction with axiom eight prohibits the use of *all* cyclic routes.

Axiom five can be seen as a definedness condition on the Send action, the literal reading is "if a message m is not currently at node n then it is not permitted to send it somewhere, from node n " the alternative contrapositive reading, "if it is permitted for a message m to be sent from node n to node n' then the message must currently be at node n " is also intuitive. Although equivalent the former reading is perhaps more intuitive since it spells out the fact that a message cannot be sent anywhere unless it is at the node from which it is to be sent.

The next axiom describes how the priority system operates. Priorities are organised in a nested fashion in the following way: any urgent message is also considered to be an important message and any important message is also a message of normal priority. Both these conditions are represented explicitly by the two conjuncts of axiom six. Further, due to the transitive nature of material implication, we can also deduce that an urgent message is also one of normal priority. The nesting of priorities is important because, as we shall soon see, all other things being equal, it means that routes available to a message of high priority necessarily include all the routes available to messages of lower priority. The reverse is not true.

Description of actions.

The "Enter" action is described in axiom seven; it is an unconditional description and when successfully performed the "Enter" action leads to a scenario in which the following properties hold: firstly the message being entered is recorded as being at the node at which it was entered, secondly the node of entry is labelled as the origin of the message and finally permission to enter the same message into any node in the system is prohibited. The first two effects of this actions should be self explanatory, the third is required to ensure that no copies of the message are re-entered into the network as this will violate the uniqueness condition of axiom one.

Similar to the "Enter" action, the "Display" action also has a non conditional description; whenever it is performed it is assumed that the message displayed has been received - we do not describe this process fully because it does not affect our primary concern, which is the routing of messages - hence its only effect is to permit the re-entry of the displayed message into the network. To some extent "Enter" and "Display" are complementary actions, the former governing how messages are entered into the network and the latter how they leave. In particular the "Enter" action prohibits further entries of a message already in the network whereas the "Display" action derogates this prohibition after the message in question leaves the system.

The next three axioms deal with the description of the "Send" action; we shall look at each of these in turn. As we said earlier the first of these, axiom seven, taken in conjunction with axiom four, is intended to eliminate cyclic routes; this means that no message can visit the same node twice while being delivered to its destination. The axiom reads as follows "sending a message from one node to another always results in a scenario from which there can be no permissible sequence of actions that results in that

same message being returned to the node from which it has just been sent". This is a little easier to see if we rewrite

$$(8) \quad \forall m \forall n, n' [\text{Send}(m, n, n')] \neg \text{Sper}(\text{At}(m) = n)$$

as

$$(8) \quad \forall m \forall n, n' [\text{Send}(m, n, n')] \forall \sigma (\text{PS}(\sigma) \rightarrow (\neg \llbracket \sigma \rrbracket \text{At}(m) = n))$$

this says that all permissible sequences are ones which do not result in the message returning to the node from which it has just been sent. Note also that because an obligation to perform an action implies permission to perform it, this axiom also precludes those sequences of actions which include obliged actions that result in the same node being arrived at. Coupling these facts with axiom four - which prohibits messages to be sent from any node to itself - we have effectively ensured that all correct implementations of this specification must contain cycle free scheduling strategies.

Whereas axiom nine is an example of how an action description can be used to express a desirable system property the next two axioms are more to do with describing the effects of the "Send" action itself rather than as a means to express some global system property. Axiom ten reads as " if a message which is currently at a node n is sent to node n' and n' does not happen to be the destination of the message then the status of where the message resides is changed from being at n to being at n' ". This should not need further explanation.

Axiom eleven is similar to nine except that it deals with the case where the message is in fact sent to its destination node. The action of sending a message m from a node n to its destination n' results in a scenario in which the the message m is not present at any node in the network and the existence of a permissible sequence of actions that leads to an opportunity (permission) for the system to display the message m at its destination node. Note that it is not sufficient to simply assert that the message is no longer at node n' since this does not rule out the possibility of it being elsewhere. Hence we must ensure that messages arriving at their destination nodes leave the system by stating that the message in question is not at any node; we can see this more clearly in the following rewrite of axiom eleven,

$$(11) \quad \forall m \forall n, n' \quad \text{DES}(m, n') \rightarrow \\ [\text{Send}(m, n, n')] \forall n'' (\neg \text{At}(m, n'') \wedge \text{Sper}(\text{P}(\text{Disp}(m, n'')))).$$

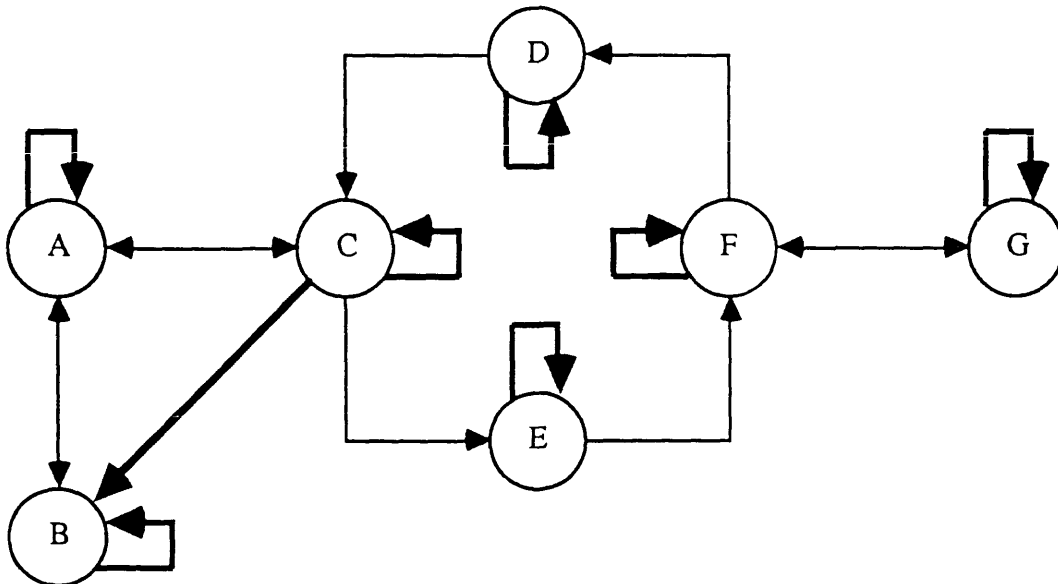
The second effect of sending a message to its destination node is that the system is always permitted some sequence of action which when performed results in the message being (obliged to be) displayed at its destination node. This does not mean that the message will be displayed just that the opportunity must exist that it will. At first sight this may seem a little too loose to be of any use but the reader must bear in mind that we are primarily concerned with how messages can be routed in the network and not how messages are displayed once they reach their destination nodes. If we enlarged the scope of the specification to deal with when and how messages are to be displayed this condition may then be strengthened.

Normal routes.

The routes available to messages of normal priority are characterised by the conjunction of properties expressed in axiom twelve. The general form of each of these properties is

$$(At(m,n) \rightarrow P(Send(m,n,n')))$$

where m is any message and n and n' are node constants, it should be read as if a message is at node n then it is permitted to be sent from node n to node n' . We can portray the content of axiom twelve more clearly by the following diagram.



Here, the thin edges represent the labelled nodes of the network and the directional arcs the permissible routes for normal messages. For instance the arc from node F to node D

means that any message of normal priority currently at node F is permitted to be sent to node D. The use of a diagram can also be used to show the difference between no permission to do something and an absence of any permission. The broad edges represent prohibitions and anything else can be considered as open (either prohibited or permitted) to either refinement or implementation decisions. Further because messages of higher priority are also considered to be normal messages routes prescribed for higher priorities must respect the prohibitions imposed at this level and may only extend positively the availability of routes.

Important routes.

The next four axioms define how the routes available to normal messages may be extended if the message is considered important. The first of these, axiom thirteen is straightforward enough; it simply adds the possibility of important messages being sent directly from node E to node D. This is a fairly useful addition since messages from B, A, C or E *may* all bypass node F whenever they are destined for node D.

The next axiom is slightly more complicated (due mainly to the use of the nondeterministic combinator +); the essence of this axiom is that it prescribes a choice between two potentially new additions to the routes available to important messages. Intuitively it states that whenever an important message is at node C it is allowed to go to node F either by performing the sequential composition of being sent first to E and then to F or else directly from node C to F. Remember that the nondeterministic action

$$(\text{Send}(m,C,E) ; \text{Send}(m,E,F)) + (\text{Send}(m,C,F))$$

is one which must either behave like the action $(\text{Send}(m,C,E) ; \text{Send}(m,E,F))$ or like the action $(\text{Send}(m,C,F))$. Hence either the message in question goes from C to F via E or it goes directly there. Since this nondeterministic action is permitted whenever an important message is at node C an implementation of this specification must have one of the above routes available. Compare this to the expression

$$P(\text{Send}(m,C,E) ; \text{Send}(m,E,F)) \wedge P(\text{Send}(m,C,F))$$

which forces all correct implementations of our Network to make provision for both routes. Note that the possibility of constructing our network with both routes is not ruled out by the expression involving the nondeterministic combinator but neither is it a requirement - as is the case in the latter expression.

The aim of the next two expressions in Netspec are very similar and we shall deal with them as a pair. The first, axiom fifteen, states that all important messages which are entered into our network at node A are permitted to be sent to either node D or node E whilst the second, axiom sixteen, states that all important messages entering our network at node G are also either permitted to be sent to node D or node E. There are of course differences between the two expressions; in the former the choice is expressed as a permitted nondeterministic action and in the latter as a disjunction of permitted properties. Given our exposition of axiom fourteen the reader should have little difficulty in understanding the implications of being permitted to do the nondeterministic action so we shall skip this and look at axiom sixteen. Remembering that $\text{Per}(\phi)$ is defined as the existence of at least one permitted action which has as effect ϕ , then axiom sixteen requires that for all important messages entered into the network at node G there exist at least one permitted action which has either the effect of sending a message to node D - represented by the property $\text{At}(m,D)$ - or the effect of sending the message to node E - represented by $\text{At}(m,E)$. Note that because of axiom one of Netspec it is not possible for any action to fulfill both these properties at the same time. Taking all this into account we can satisfy the requirement of axiom sixteen by an expression such as

$$\forall m \text{ Prty}(m,IM) \rightarrow [\text{Enter}(m,G)] P((\text{Send}(m,G,D)) + (\text{Send}(m,G,E)))$$

but we must remember that this is only one possible way in which the this requirement can be satisfied the following would also do

$$\forall m \text{ Prty}(m,IM) \rightarrow [\text{Enter}(m,G)] P((\text{Send}(m,G,D))) \vee P((\text{Send}(m,G,E)))$$

since there would be at least one permitted action capable of resulting in one of the properties $\text{At}(m,D)$ or $\text{At}(m,E)$. It is not difficult to see that many other actions may satisfy the requirements of axiom sixteen and hence that it is a more abstract condition than the more concrete requirement of axiom fifteen.

Urgent routes.

The remainder of Netspec deals with the behaviour of urgent messages; the first of these, expressed by axiom seventeen, requires that any urgent message, upon entering the network, must be moved from its entry point as part of the network's next observable action. This is intended to represent the fact that urgent messages must be serviced immediately upon entry to the network; note that this does not mean that urgent messages are delivered upon being entered into the network just that they are moved (hopefully towards their destination) from their entry point. This is captured by the

property that entering an urgent message 'm' at node 'n' results in the obligatory property that 'm' is no longer at 'n'. Because an obligatory property ensures that there exists one and only one obligatory action which results in that property this means that any implementation of the network must, upon receipt of an urgent message, perform as its next action something which moves that urgent message. This does not mean that all next actions must be restricted to just moving that urgent message; for instance it could be that the next action performed is the parallel action of sending the urgent message somewhere and sending a normal message somewhere else. It does however require that whichever action is performed next it has as a consequence the obligatory property in question. In many ways this is much like the 'Next' operator found in most temporal logics; see for example [RU71] and [Pnu77], since this too is used to ensure that the next "state" of the system exhibits some desirable property.

In terms of routing messages, axiom eighteen is very useful indeed; its purpose is to permit (as one action) all those sequentially composed "Send" actions corresponding to all the two element permitted sequences of "Send" actions. We can see this as a form of limited transitivity on "Send" operations, it is not general transitivity since we cannot recursively apply the axiom to the new permissions - involving sequentially composed "Send" actions. A more powerful axiom would be the following

$$\forall m \forall n, n', n'' \ (\text{Prty}(m, \text{UG}) \wedge \text{At}(m, n) \wedge \text{PS}(\langle \text{Send}(m, n, n'), \text{Send}(m, n', n'') \rangle) \rightarrow \text{P}(\text{Send}(m, n, n''))).$$

In the base case this means that any two element permitted sequence of "Send" operations entails permission to perform the sequential composition of the actions in that sequence. Applying the axiom recursively, so as to include these new permissions, we see that the condition is more general than axiom eighteen and permits the (ordered) sequential composition of "Send" operations of all - not just the two element sequences - permitted sequences of "Send" operations. This means that any message permitted to be delivered as a sequence of "Send" operations may now be delivered as one operation. However this may in fact be too powerful and it is more likely that some form of local transitivity - like that of axiom eighteen, connecting the nearest neighbours of neighbouring nodes for example, would be more appropriate.

The last axiom of our specification, axiom nineteen, is quite similar to the last one except it deals with parallel actions; it states that any two distinct urgent messages which are both permitted to be sent from their present node to some other node are permitted to do so in parallel. Again the axiom is not recursive but can be made so - like axiom eighteen

- if desired. The condition (the antecedent of this axiom) ensuring messages should be distinct is needed in order to avoid sending the same message to two different nodes since this would, if allowed, violate axiom one.

Although Netspec has demonstrated the use of both primary and secondary action combinators it should not be taken as an attempt to capture the complete requirements of some intended network. For a more comprehensive treatment we would refer the reader to the recent work of [D-H86] in which the author proposes a top down stepwise refinement methodology for network protocol specification using as part of his logical basis the notions of permissible and obligatory action described here. The reader is also referred to [JKM86] wherein the authors use the framework developed in this dissertation to formally describe the OSI (Open Systems Interconnection) network service protocol, although this latter work also makes use of something we have not yet introduced - the idea of an agent. Briefly, an agent identifies a specific component of a system and rather than describing anonymous "system" actions, actions may be identified with the specific component of a system which performs the action. For instance in our network example rather than talk about an anonymous collection of system wide actions we may identify specific nodes as agents, each of which has its own abilities and behaviour. Apart from the advantages of localising action descriptions and behavioural norms to specific agents - and thereby breaking down the complexity of large system specifications, the ability to refer to the things which perform actions is often indispensable if we are to model reality directly. It is to a treatment of this concept of agent to which we now turn.

CHAPTER FIVE

AGENTS

5.1 Introduction.

The concept of an agent is now quite common amongst designers of computing based systems; its wide use has made it an acceptable term for everything ranging from a concrete hardware processing unit to an idealised fragment of reality. There is however a common thread in all these uses - an agent is identified as a non passive object, something which has the capacity to change the environment in which it exists. The notion has perhaps been most exploited in the artificial intelligence world where it is indispensable for modelling things like dialogue [Sad88] and [CL85], planning [Ped86] and various belief/knowledge/truth maintenance systems [Pol86], [FH85] and [AH85].

For our purposes the introduction of agents will allow us to attach a "doer" to our presently anonymous (system wide) concept of actions; in other words we will be able to describe actions relative to the things which perform them. This has several beneficial consequences: it models reality more accurately, it aids us in the task of structuring our specifications, it will help to localise the way in which we prescribe the behaviour of a system, and, finally, it will allow us to describe interaction, at both the structural and behavioural levels, between the identifiable active components (agents) of a system.

The first of these benefits is easily seen if we examine the unnaturalness of our previous examples. In our shipping example, for instance, we defined an action called "Sail" which had two parameters, a ship ('s', say) and a destination port ('p', say), the idea being that performing such an action would lead ship 's' to be docked at port 'p'. The emphasis here is on the phrase "performing such an action", which immediately raises the question of who/what can/should perform such an action ? The intuitive answer is obvious; sailing to a port should be an act attributed to the ship in question but this is only implicitly achieved by parameterisation of actions. Apart from wanting to make this connection between the performer and that which is performed more explicit, we also need to distinguish (a little more clearly) between agents who may be involved in an action and those which are attributed with performing that action. As an example of this consider our earlier "network" example in which messages were transferred between nodes by a "send" action. Again, there was no notion of a node sending a message, just the more anonymous idea of a message somehow being sent !! It would, we believe, be more natural to think of nodes sending messages to other nodes; again this may be achieved by defining a three parameter "send" action, where "send(n, m, n')" denotes the action of node n sending message m to node n'. But this does not explicitly

discriminate between the agent (node) sending the message and the agent (node) receiving it (although we may be able to deduce this from the way in which the effects of the action are described). We believe that the role of the *performing* agent is sufficiently important enough to factor out as a special case and should not be left as an implicit consequence to be derived from how specific actions are described; the formalism we shall present later will reflect this. Not to do so would lose us the very mechanism we are striving for - an explicit means of attaching actions to the objects which can perform them.

We have already gone some way towards describing the second benefit that the introduction of agents brings us, namely a certain amount of structuring. Relativising the ability for change in a system to a number of named agents leads in general to more manageable specifications, since rather than inspecting/describing the entire system's capacity for change we may decompose this awesome task into two stages. The first stage may be to describe each agent's capacity to perform actions in isolation (local to its own capabilities), leaving for the second stage how these agents interfere with each other by performing actions which effect other agents. Although the division between these two stages will not always be clear cut the mere fact that we have a unit smaller than the entire system on which to focus attention cannot be anything other than an aid to structuring our specifications - no matter what methodology we follow to arrive at such specifications.

If we are to follow this scheme of associating actions with the objects which perform them, then the next consequence is not so much a benefit as a commonsense strategy. If we are to describe actions relative to the agents which perform them, then it is only natural that we should also prescribe the behaviour of individual agents as well. Hence, permission and obligation should be modified to be permission and obligation for specific agents to do actions. Apart from the benefits already advanced for relativising the description of actions to agents, this also provides us with another dimension by which we can weaken the force of obligations. This is because an agent specific obligation will not suspend any permission to do an action not specific to the agent in question and consequently allows other agents to behave as normal.

As an illustration of this, we may return once again to our shipping example of previous chapters. There we found that an unfortunate consequence of one ship incurring an obligation to unload timber meant that all other ships could not perform any actions until the obligated ship had unloaded all its timber. In addition to this it was also noted that

only one obligation could be outstanding at any one time, a particularly surreal constraint given that it is quite probable that in reality one ship might be obliged to unload timber whilst another was obliged to sail to some destination. As we shall see, relativising the deontic operators to specific agents deals with both these anomalies. Note that this does not mean that we have weakened the force of permissions and obligations *per se* but rather that we have localised their previously more general scope. The reader is also asked to note that this localising of effect would not be possible without an explicit discrimination between the role of an agent as a performer of an action.

The last benefit in our list has already been touched on when we talked about how the introduction of agents helped us to structure specifications. Here, we pointed out that in addition to helping us focus on describing and prescribing actions relative to smaller units, this would also be helpful in describing/prescribing the interaction between these smaller units. The "network" example of the previous chapter is a good instance of where this might have been useful. Imagine that the agents in this application domain are nodes - it is quite likely that, depending on the way in which the transmission medium between the nodes is arranged and, of course, the protocol of communication between nodes, some nodes may need to request "neighbouring" nodes to pass on certain messages to their neighbours. This being the case it is quite likely that neighbouring nodes will engage in a certain amount of interaction in order to convey a message to its destination. Such interaction can occur at two levels, at a structural level where the effects of one agent doing an action in response to another agent's action can be described and at a prescriptive level where one agent causes another to incur an obligation or be granted/revoked a certain permission. We shall see examples of both after we have introduced the technical details of how we propose to accommodate this notion of agent.

Given all the above advantages the reader may be wondering why the concept of an agent was not incorporated into our notion of action description and prescription right from the start. The major reason can be traced back to the introduction of this work where we remarked upon the modularity criteria of a requirement specification formalism. We are firm believers in starting out with a general kernel framework which can then be extended in many dimensions; we hope that this belief has been borne out in the writing of this dissertation, where each chapter extends (in some way) the expressive powers of the previous one. The decision to incorporate the agent extension last was based on three observations: firstly, our original characterisation of action can always be thought of as the special one agent case of the more general situation described in this

chapter; secondly, the extra model constraints associated with the more general case would have unnecessarily clouded explanations involving the deontic and more general model constraints; and last, but by no means least, generalising to the more general case last provides us with an opportunity to marshal together all the fragmented extensions developed in this dissertation into one formal description of our formalism.

The remainder of this chapter is structured in three sections, the first of which will be used to present a final version of our logic, the second will describe some of the specification issues involved in the introduction of agents and finally the third section will provide an example of a specification in this logic.

5.2 DL (*Deontic Logic*).

The purpose of this section is to present the final version of the logic we have been developing throughout this dissertation. We have decided, rather egotistically, to call the resulting formalism "Deontic Logic" or DL for short; as usual, we shall proceed in two stages, giving first the language and then the axioms of DL. As before, the reader is asked to refer to this axiomatisation of DL in conjunction with the explanation that follows it.

LANGUAGE

Syntactic Categories.

- A non empty finite collection of sorts $S = \text{Act} \cup \text{Agt} \cup S(\text{Act}) \cup \{s_1, s_2, \dots\}$, where: **Act** denotes the action sort, **Agt** represents the agent sort, and $S(\text{Act})$ denotes a non empty sort populated by non null finite sequences of actions.
- Constant symbols:
 - Logical: $\mathbf{n}$.
 - Extra-logical: for each sort $s \in S$ there is a set (possibly empty) of constant symbols each of which is said to be of sort s .
- Action names: for each $n > 0$ and each n -tuple, $\langle s_1 \dots s_n \rangle$ such that $s_1 \in S, \dots s_n \in \text{Act}$, there is a set (possibly empty) of n -place action names each of which is said to be of type $\langle s_1 \dots s_n \rangle$.
- Predicate symbols:
 - Logical:
P of type **Act**, O of type **Act**, Pref of type **Act** and OS of type $S(\text{Act})$.
 - Extra-logical:
for each $n > 0$ and each n -tuple, $\langle s_1 \dots s_n \rangle$ such that $s_1 \in S, \dots s_n \in S$, there is a set (possibly empty) of n -place predicate symbols each of which is

said to be of type $\langle s_1 \dots s_n \rangle$.

- Function symbols:

- Logical:

; of type $\langle \text{Act}, \text{Act}, \text{Act} \rangle$ $\langle \rangle$ of type $\langle \text{Act}, S(\text{Act}) \rangle$
 $\parallel$ of type $\langle \text{Act}, \text{Act}, \text{Act} \rangle$ $,$ of type $\langle S(\text{Act}), \text{Act}, S(\text{Act}) \rangle$
 $+$ of type $\langle \text{Act}, \text{Act}, \text{Act} \rangle$

- Extra-logical:

for each $n > 0$ and each $n+1$ -tuple, $\langle s_1 \dots s_n, s_{n+1} \rangle$ such that $s_1 \in S, \dots, s_{n+1} \in S$, there is a set (possibly empty) of n -place function symbols each of which is said to be of type $\langle s_1 \dots s_{n+1} \rangle$.

- Equality symbols: for certain (possibly all) sorts $s \in S$ there may be a special predicate symbol $=_s$ of sort $\langle s, s \rangle$ representing equality between objects of sort s .
- Variables: the usual infinite collection of distinct variables for each sort $s \in S$.
- Quantifiers: for each sort $s \in S$ we have the universal quantifier $\forall_s$ and the existential quantifier $\exists_s$.
- Logical operators:
 $\neg, \leftrightarrow, \rightarrow, \wedge, \vee,$
 $[_, _], \llbracket _, _ \rrbracket,$
 $\text{Per}(_, _), \text{Obl}(_, _), \text{Sper}(_, _)$ and $\text{Sobl}(_, _).$
- Punctuation: $(,),$ and $,.$

Formation Rules.

- Terms:

- for each sort $s \in S$ a variable or constant of sort s is a term.
- if $t_1 \dots t_n$ are terms of sorts $s_1 \dots s_n$ respectively and $s_1 \dots s_n$ are all taken from S

and f is a function symbol of type $\langle s_1 \dots s_n, s_{n+1} \rangle$ then $f(t_1 \dots t_n)$ is a term of sort s_{n+1} .

- if $t_1 \dots t_n$ are terms of sorts $s_1 \dots s_n$ respectively and $s_1 \dots s_n$ are all taken from S and a is an action symbol of type $\langle s_1 \dots s_n \rangle$ then $a(t_1 \dots t_n)$ is a term of sort **Act**.

- nothing else is a term.

- **Atoms:**

- if $t_1 \dots t_n$ are terms of sorts $s_1 \dots s_n$ respectively and $s_1 \dots s_n$ are all taken from S and p is a predicate symbol of type $\langle s_1 \dots s_n \rangle$ then $p(t_1 \dots t_n)$ is an atom.

- for each sort $s \in S$, given two terms of s , t_1 and t_2 , $t_1 =_s t_2$ is an atom, provided that $=_s$ is in the language.

- nothing else is an atom.

- **Formulae:**

- An atom is a formula.

- The logical constant **$\perp$** is a formula.

- If ϕ is a formula so is $\neg \phi$.

- If ϕ and ψ are formulae then $(\phi \vee \psi)$, $(\phi \wedge \psi)$, $(\phi \rightarrow \psi)$ and $(\phi \leftrightarrow \psi)$ are also formulae.

- If α is a term of sort **Act**, A is a term of sort **Agt** and ϕ is a formula, then $[A, \alpha] \phi$ is a formula.

- If σ is a term of sort $S(\mathbf{Act})$, A is a term of sort **Agt** and ϕ is a formula then $\llbracket A, \sigma \rrbracket \phi$ is a formula.

- If φ is a formula and A is a term of sort **Agt** then $\text{Per}(A, \varphi)$, $\text{Obl}(A, \varphi)$, $\text{Sper}(A, \varphi)$ and $\text{Sobl}(A, \varphi)$ are formulae.
- For each sort $s \in \mathbf{S}$, if x is a variable of sort s and φ is a formula then $\forall x \varphi$ and $\exists x \varphi$ are formulae.
- nothing else is a formula.

Axioms.

If φ, ψ and τ are formulae, A a term of sort **Agt**, α is a term of sort **Act** and σ is a term of sort $\mathbf{S}(\mathbf{Act})$ then the following are axioms.

1. $\varphi \rightarrow (\psi \rightarrow \varphi)$
2. $(\tau \rightarrow (\varphi \rightarrow \psi)) \rightarrow ((\tau \rightarrow \varphi) \rightarrow (\tau \rightarrow \psi))$
3. $(\neg \varphi \rightarrow \neg \psi) \rightarrow ((\neg \varphi \rightarrow \psi) \rightarrow \varphi)$
4. $\forall x \varphi(x) \rightarrow \varphi(t)$ where t is free for x in φ
5. $(\forall x (\varphi \rightarrow \psi)) \rightarrow (\varphi \rightarrow \forall x \psi)$ where x is not free in φ
6. $[A, \alpha] \text{TRUE}$
7. $([A, \alpha] (\varphi \rightarrow \psi)) \rightarrow (([A, \alpha] \varphi) \rightarrow ([A, \alpha] \psi))$ $\varphi \rightarrow \psi \rightarrow \varphi \rightarrow \psi$
8. $([A, \alpha] \neg \varphi) \rightarrow (\neg [A, \alpha] \varphi)$ $\neg \varphi \rightarrow \neg \varphi$
9. $\forall x ([A, \alpha] \varphi) \leftrightarrow ([A, \alpha] \forall x \varphi)$ where x is not free in α or A $\forall x \varphi \leftrightarrow \forall x \varphi$
10. $\exists x ([A, \alpha] \varphi) \rightarrow ([A, \alpha] \exists x \varphi)$ where x is not free in α or A $\exists x \varphi \rightarrow \exists x \varphi$
11. $(([A, \alpha] \varphi) \wedge ([A, \alpha] \psi)) \leftrightarrow ([A, \alpha] \varphi \wedge \psi)$ $\varphi \wedge \psi \leftrightarrow \varphi \wedge \psi$
12. $(([A, \alpha] \varphi) \vee ([A, \alpha] \psi)) \rightarrow ([A, \alpha] \varphi \vee \psi)$ $\varphi \vee \psi \rightarrow \varphi \vee \psi$
13. $\llbracket A, < \alpha > \rrbracket \varphi \leftrightarrow [A, \alpha] \varphi$ $\varphi \leftrightarrow \varphi$
14. $\llbracket A, < \sigma, \alpha > \rrbracket \varphi \leftrightarrow \llbracket A, \sigma \rrbracket [A, \alpha] \varphi$ $\varphi \leftrightarrow \varphi$
15. $\mathbf{n} \rightarrow (P(A, \alpha) \leftrightarrow [A, \alpha] \mathbf{n})$ $\text{variable } n \rightarrow [A, \alpha] n$ $n \leftrightarrow n$
16. $\neg [A, \alpha] \mathbf{n} \rightarrow [A, \alpha] \neg \mathbf{n}$ $\neg n \rightarrow \neg n$
17. $\text{Pref}(A, \alpha) \leftrightarrow \exists \beta (\neg (\beta = \alpha) \wedge P(A, \beta))$
18. $O(A, \alpha) \rightarrow P(A, \alpha)$

19. $O(A, \alpha) \rightarrow \neg \text{Pref}(A, \alpha)$
 20. $\text{PS}(A, \langle \sigma, \alpha \rangle) \leftrightarrow (\text{PS}(A, \sigma) \wedge \llbracket A, \sigma \rrbracket P(A, \alpha))$
 21. $\text{PS}(A, \langle \alpha \rangle) \leftrightarrow P(A, \alpha)$
 22. $\text{OS}(A, \langle \alpha \rangle) \leftrightarrow O(A, \alpha)$
 23. $\text{OS}(A, \langle \sigma, \alpha \rangle) \leftrightarrow (\text{OS}(A, \sigma) \wedge \llbracket A, \sigma \rrbracket O(A, \alpha))$
 24. $P(A, \alpha; \beta) \rightarrow ([A, \alpha; \beta] \varphi \leftrightarrow [A, \alpha][A, \beta] \varphi)$
 25. $P(A, \alpha \parallel \beta) \rightarrow ([A, \alpha] \varphi \wedge [A, \beta] \psi \rightarrow ([A, \alpha \parallel \beta] \varphi \wedge \psi))$
 26. $\alpha \parallel \beta =_{\text{Act}} \beta \parallel \alpha$
 27. $P(A, \alpha + \beta) \rightarrow ([A, \alpha] \varphi \wedge [A, \beta] \psi \rightarrow ([A, \alpha + \beta] \varphi \vee \psi))$
 28. $([A, \alpha] \varphi \wedge [A, \beta] \psi \wedge \neg [A, \alpha] \psi \wedge \neg [A, \beta] \varphi) \rightarrow (\neg [A, \alpha + \beta] \varphi \wedge \psi)$
 29. $\alpha + \beta =_{\text{Act}} \beta + \alpha$ $\psi \wedge \psi \wedge \neg \psi \wedge \neg \psi \rightarrow \neg(\psi \wedge \psi)$
 30. $\text{Per}(A, \varphi) \leftrightarrow \exists \alpha [A, \alpha] \varphi \wedge P(A, \alpha)$
 31. $\text{Obl}(A, \varphi) \leftrightarrow \exists \alpha [A, \alpha] \varphi \wedge O(A, \alpha)$
 32. $\text{Sper}(A, \varphi) \leftrightarrow \exists \sigma \llbracket A, \sigma \rrbracket \varphi \wedge \text{PS}(A, \sigma)$
 33. $\text{Sobl}(A, \varphi) \leftrightarrow \exists \sigma \llbracket A, \sigma \rrbracket \varphi \wedge \text{OS}(A, \sigma)$
- EQ $\forall x, y ((\varphi(x) \wedge x=y) \rightarrow \varphi(y))$ where y is not within the scope of a modal operator.

DL Models.

Because of the possibility of interaction between agents we shall need to introduce an auxilliary notion of obligatory sequence in order to capture the dynamic aspects of obligations properly. To define this "model" level notion of obligation we will require the following logical machinery; as before we divide this into additional language and additional axioms.

Language.

Syntactic Categories.

- Sorts: **AAP**, a non empty collection of agent - action pairs and **AS(Act)**, a non empty collection of non null finite agent - action pair sequences.
- Constants: For both these additional sorts.
- Variables:
 - The usual infinite collection of variables for sort **AAP**, π, π', π''
 - The usual infinite collection of variables for sort **AS(Act)**, $\sigma, \sigma', \sigma''$
- Logical functions:
 - $\langle _ , _ \rangle$ of type $\langle \text{Agt}, \text{Act}, \text{AAP} \rangle$
 - $\langle \rangle$ of type $\langle \text{AAP}, \text{AS(Act)} \rangle$
 - $::$ of type $\langle \text{AS(Act)}, \text{AAP}, \text{AS(Act)} \rangle$
- Logical predicates:
 - OS** of type $\langle \text{Agt}, \text{AS(Act)} \rangle$
 - MOS** of type $\langle \text{Agt}, \text{AS(Act)} \rangle$
- Logical operator: $\llbracket _ \rrbracket$

Formation rules.

- If σ is of type **AS(Act)** and ϕ is a formula then $\llbracket \sigma \rrbracket \phi$.
- All previous (standard) clauses apply.

Axioms.

- M1. $\llbracket \langle \langle A, \alpha \rangle \rangle \rrbracket \phi \leftrightarrow [A, \alpha] \phi$
- M2. $\llbracket \sigma :: \langle A, \alpha \rangle \rrbracket \phi \leftrightarrow \llbracket \sigma \rrbracket [A, \alpha] \phi$

$$M3. \quad OS(A, \langle \langle B, \beta \rangle \rangle) \leftrightarrow (\exists \alpha [B, \beta] O(A, \alpha))$$

$$M4. \quad OS(A, \sigma :: \langle B, \beta \rangle) \leftrightarrow (OS(A, \sigma) \wedge \exists \alpha (\llbracket \sigma :: \langle B, \beta \rangle \rrbracket O(A, \alpha)))$$

$$M5. \quad MOS(A, \sigma :: \langle B, \beta \rangle)$$

$$\leftrightarrow (OS(A, \sigma) \wedge \neg \exists \alpha (\llbracket \sigma :: \langle B, \beta \rangle \rrbracket O(A, \alpha)))$$

The language should be quite easy to follow, we have introduced two new sorts **AAP** and **AS(Act)**, the former of which defines the universe of agent - action pairs and the latter the universe consisting of non null finite sequences of agent - action pairs. We may refer to elements of **AAP** via the logical function $\langle \rangle$ which accepts an agent and an action and returns the corresponding pair; the usual pairing axioms do apply but we shall not mention them for the sake of brevity and clarity. Elements of the **AS(Act)** sort may be constructed using the two functions $\langle \rangle$ and $::$, the first of which constructs the one element agent - action pair sequence, whereas the second is used to construct larger sequences by concatenating agent - action pairs to existing sequences.

The logical operator $\llbracket _ \rrbracket$ extends the usual operator for describing the effects of a single agent performing a sequence of actions by describing the effects of performing a sequence of actions where each action may be performed by a different agent. Hence given the following agent - action sequence

$$\langle \langle A, \alpha \rangle \rangle :: \langle \langle B, \beta \rangle \rangle :: \langle \langle C, \kappa \rangle \rangle :: \langle \langle D, \delta \rangle \rangle$$

asserting

$$\llbracket \langle \langle A, \alpha \rangle \rangle :: \langle \langle B, \beta \rangle \rangle :: \langle \langle C, \kappa \rangle \rangle :: \langle \langle D, \delta \rangle \rangle \rrbracket \varphi$$

is equivalent to asserting

$$[A, \alpha][B, \beta][C, \kappa][D, \delta] \varphi.$$

A recursive scheme to this effect is given by axioms M1 and M2 above. Using this more general notion of sequence we can then define a slightly weaker notion of obligation **OS** and maximal obligation **MOS** sequence. The former is defined by axioms M3 and M4 and defines an obligation sequence **OS** for an agent *A*, to be any sequence of agent - action pairs such that after each element of the sequence is performed the agent *A* has some action which it is obliged to perform. A maximal obligation sequence, **MOS**, for an agent *A* is then defined, by axiom M5, as the smallest obligation sequence for that agent.

We shall see the use of these looser definitions of sequences later when we discuss how

the introduction of agents has altered our previous definition of model. The reader is asked to consult this discussion whilst reading the model definition below. We may now define models for DL as follows:

Model Structures.

A *model structure* for a DL specificand **sp** is given by a pair $\{ \mathcal{D}, \mathcal{F} \}$ where

$\mathcal{D}$: is a set of deontic scenarios, where a scenario is a theory extension of the specificand **sp**.

$\mathcal{F}$: is a collection of typed functions, one for each element of the sort **AAP**, where the type of such functions is

$$f_{\langle A, \alpha \rangle} : \text{Agt} \times \mathbf{t}(\alpha) \times \mathcal{D} \rightarrow \mathcal{D}$$

where $\mathbf{t}(\alpha)$ denotes the type of the action name α , **Agt** denotes the agent sort and $\mathcal{D}$ is the domain of potential scenarios for the specification **sp**.

Models.

A *model structure* $\{ \mathcal{D}, \mathcal{F} \}$ for **sp** is a *model* for **sp** iff the following conditions are met,

- $\forall s \in \mathcal{D} \quad \forall A \in \text{Agt} \quad \forall \alpha \in \text{Act}$
 $\text{if } [A, \alpha] \varphi \in \text{TH}(s) \text{ then } \varphi \in \text{TH}(f_{\langle A, \alpha \rangle}(s))$
- $\forall s \in \mathcal{D} \quad \forall A \in \text{Agt} \quad \forall \alpha \in \text{Act}$
 $\text{if } \neg [A, \alpha] \varphi \in \text{TH}(s) \text{ then } \varphi \notin \text{TH}(f_{\langle A, \alpha \rangle}(s))$
- $\forall s \in \mathcal{D}, \forall A, B \in \text{Agt} \quad \forall \alpha, \beta \in \text{Act}$
 $\text{if } [A, \alpha] O(B, \beta) \notin \text{TH}(s) \text{ then } O(B, \beta) \notin \text{TH}(f_{\langle A, \alpha \rangle}(s))$
- $\forall s \in \mathcal{D}, \forall A, B \in \text{Agt} \quad \forall \alpha, \beta \in \text{Act}$
 $\text{if } A \neq B \text{ and } O(B, \beta) \in \text{TH}(s) \text{ and } [A, \alpha] \neg O(B, \beta) \notin \text{TH}(s)$
 $\text{then } [A, \alpha] O(B, \beta) \in \text{TH}(s)$

- $\forall s \in \mathcal{D}, \forall A, B \in \text{Agt}, \forall \alpha, \beta, \delta \in \text{Act}, \forall \sigma \in \text{AS}(\text{Act}),$
if $P(A, \alpha) \in \text{TH}(s)$ and $O(A, \delta) \notin \text{TH}(s)$
and $\text{MOS}(A, \sigma) \in \text{TH}(f_{\langle B, \beta \rangle}(s))$ and $\neg P(A, \alpha) \notin \text{TH}(f_{\sigma} f_{\langle B, \beta \rangle}(s))$
then $P(A, \alpha) \in \text{TH}(s) \leftrightarrow P(A, \alpha) \in \text{TH}(f_{\sigma} f_{\langle B, \beta \rangle}(s)).$
- $\forall s \in \mathcal{D}, \forall A, B \in \text{Agt}, \forall \alpha, \beta, \delta \in \text{Act}, \forall \sigma \in \text{AS}(\text{Act}),$
if $\neg P(A, \alpha) \in \text{TH}(s)$ and $O(A, \delta) \notin \text{TH}(s)$
and $\text{MOS}(A, \sigma) \in \text{TH}(f_{\langle B, \beta \rangle}(s))$ and $P(A, \alpha) \notin \text{TH}(f_{\sigma} f_{\langle B, \beta \rangle}(s))$
then $\neg P(A, \alpha) \in \text{TH}(s) \leftrightarrow \neg P(A, \alpha) \in \text{TH}(f_{\sigma} f_{\langle B, \beta \rangle}(s)).$

As before we have used $\text{TH}(s)$ to denote the theory (collection of formulae closed under logical consequence) corresponding to scenario (the theory presentation) s and $\in$ and $\notin$ to indicate whether or not a formula belongs to a theory. The use of f_{σ} is introduced as a shorthand for the sequence of functions corresponding to the agent - action pair sequence σ . The definition of these functions is as follows

$$f_{\sigma} =_{\text{def}} f_{\langle A, \alpha \rangle}^n \circ f_{\langle A, \alpha \rangle}^{n-1} \circ \dots \circ f_{\langle A, \alpha \rangle}^1$$

where σ is the sequence, $\langle \langle A_1, a_1 \rangle \rangle :: \dots :: \langle A_n, a_n \rangle$, $f_{\langle A, \alpha \rangle}^i$ denotes the function corresponding (as defined above) to the agent - action pair $\langle A_i, a_i \rangle$ and $\circ$ denotes functional composition.

This concludes our presentation of the syntax and semantics of DL; in the next section we shall be concerned with providing an intuitive account of this syntax and semantics. In order to be concise we shall only really discuss how the introduction of agents has changed CDAL syntax and semantics.

5.3 DL Specifications.

The purpose of this section is to explore what impact the introduction of agents has had on the creation of specifications. The first thing to note is the introduction of a new syntactic category, **Agt**; in many respects elements of this category behave much like those of the action category **Act**. They can be taken/given as parameters/results of functions, be predicated upon, etcetera, yet also play a special role in the scope of modal or deontic operators. In order to provide a little more structure to the explanations that follow the remainder of this section is split into a number of parts dealing with how agents effect: action descriptions, the deontic operators, sequences of actions, action combinators and, lastly, how they change the definitions of models. Note that the first order classical logic basis of our formalism has not changed.

Action descriptions.

The most important role (from which all the others derive) that agents play in our logic is in the context of describing actions; this role has already been identified as fulfilling the part of an actor/performer/doer of actions. Hence, whereas before we described actions by formulae of the form $[\alpha] \phi$ we now write formulae such as $[A, \alpha] \phi$.

The former being read as "after the action α has been performed the property ϕ holds" the latter should be read as "after the agent A has performed the action α the property ϕ holds". This change is reflected in our models by assigning theory transforming functions to elements of the cartesian product of the **Act** and **Agt** categories, rather than to just actions. But since such functions are total this means that all models of a DL specification are required to possess one function for each of the elements in the cartesian product of **Act** and **Agt**.

At first this might seem a little strange since it means that all agents are, potentially at least, capable of performing all the actions which are elements of **Act**. Reality is simply not like this; some actions are defined for certain agents and not for others. It would be unusual for instance to admit the possibility of the memory management unit of a computer performing actions of heavy number crunching - more usually associated with the arithmetic unit. We could achieve a closer association between agent and action by giving up the flexibility of the two distinct categories **Act** and **Agt**, and entertain only a single category of specially distinguished actions but this would suffer from the same

objections, to do with agents as parameters to actions, that we raised in the introduction of this chapter.

The picture is not as bad as we have painted; under our present approach actions which are intrinsically incompatible with certain agents will not be described at all and so will be deemed to be completely underdetermined. This means that particular models will be able to choose *completely* what effects actions have, and depending on how we implement the intended system, intrinsically incompatible actions may be constructed as resulting in non normative affairs, that is we could forbid such actions occurring. Indeed this option is also open to us at the specification level where we could prohibit (by not permitting) certain agents from performing certain actions and so force all models to label such actions as leading to non normative scenarios when performed by the agent in question.

The interaction between action descriptions and the logical connectives remains the same and axioms six to twelve (inclusive) differ to those we gave in SFOAL only by the agent indexical on actions. Note that the axioms are valid only for a uniform substitution of agent, so something like $(([A,\alpha] \phi) \vee ([B,\alpha] \psi)) \rightarrow ([A,\alpha] \phi \vee \psi)$ is not valid. Again this is due to the fact that the same action performed by different agents can have different effects; the "login" action on most computer systems leads to many different effects (authorisation to various files and resources) depending on who is logging in - usually identified by a password of some kind. Hence, although in isolation, we tend to think of the "login" command of most computers as a single action it is actually a multiplicity of actions, almost certainly distinguished by who is trying to login and perhaps also by some other parameters.

In addition to being able to describe the effects of actions relative to which agents perform them this simple extension also allows us to state structural information about how agents interact with each other. Consider, for example, an expression of the form

$$[A,a] (p \rightarrow [B,b] q)$$

which says that if agent A performs action a and this results in property p then if agent B performs action b immediately after agent A has performed a then the property q will hold. This is clearer if we rewrite the above (using axiom seven) as

$$([A,a] p) \rightarrow ([A,a] [B,b] q).$$

It is the consequent of this conditional which is interesting from the aspect of describing

agent interaction. Iterated modalities of this form allow us state the structural effects of certain sequences of actions, performed in general by different agents. Examples of this may be found in our earlier shipping example, if we treat ships as agents and modify the "Sail" action to be parameterised by only the destination port.

Since all the remaining operators, such as action sequences and the deontic operators P(ermision) and O(bligation) etc, are characterised in terms of action descriptions their new definitions should now be straightforward to understand. Even so, for the sake of completeness and the opportunity to comment on their usefulness, we shall take a brief look at them.

Permission and obligation.

Considering that actions are now no longer the anonymous objects that they once were, it would be nonsensical to permit or oblige them anonymously. It seems sensible then to extend, in line with the modal operator, the deontic operators P and O so that we are able to permit or oblige actions for specific agents. The first thing to note is that permission to perform an action α for an agent A is defined, by axiom fifteen above, as (at least for all normative scenarios) the action α performed by A results in a normative scenario. The rationale for this as with axioms sixteen to nineteen is the same as in the case with no agents.

Hence, axiom sixteen (together with axiom seven) says that any action performed by any agent leads to either a normative or a non normative scenario. Axiom seventeen defines an agent refraining from an action as the property that that agent is permitted to perform some alternative action. Lastly, the static aspects of obligation are defined by axioms eighteen and nineteen; these say that an obligation to do an action by a certain agent entails that that agent is permitted to do the obliged action and not permitted to do anything else. An interesting consequence of this is that the transitory effects of an obligation are localised to the agent incurring the obligation, which means that other agents in the system may still be permitted to do other actions. This does complicate, however, the dynamic aspects of obligations but we shall delay discussing why this complication arises until we come to discuss models for specifications written in DL.

Being able to prescribe the behaviour of agents by saying which actions they are permitted or obliged to perform also helps to prescribe the interaction between agents. This is especially true if we consider that we can describe the effect of one agent's action

as granting or revoking the permission for, or even obliging, another agent to perform a certain action. As an illustration of this, consider a company database (partially) modelled by the following assertions:

- i $\forall e \text{ [Personnel-Manager, Hire}(e)]$
 $(O(\text{Accounts-Manager, Payroll}(e)) \wedge O(\text{Stock-Manager, Supply-Resources}(e)))$
- ii $\forall e, e' \text{ P}(e, \text{Fire}(e')) \leftrightarrow \text{Manager}(e, e')$
- iii $\forall e, e', e'' \text{ Manager}(e'', e) \rightarrow [e, \text{Fire}(e')] \text{ P}(e', \text{Appeal}(e''))$
- iv $\forall e, e', e'' [e, \text{Fire}(e')] [e', \text{Appeal}(e'')]$
 $(\text{P}(e'', \text{Discipline}(e)) \wedge \text{P}(e'', \text{Discipline}(e')))$.

Where these expressions are intended to be interpreted as the following English statements:

- i Whenever an employee is hired by the personnel manager the accounts manager incurs an obligation to place that employee on the company's payroll and the manager in charge of resources is obliged to supply the necessary resources that the new employee will need.
- ii An employee can fire another employee if and only if (s)he is his/her manager.
- iii If an employee is fired he has the automatic right of appeal to the manager of the employee who has fired him.
- iv Given that an appeal is made in the circumstances given in iii, then the employee to whom the appeal is being made is granted permission to discipline either the employee being fired (perhaps even by upholding the original notice to quit) or the employee who initiated this original ruling.

Note that we have not described the more structural aspects of hiring, firing and disciplining an employee, but focused instead on the more prescriptive aspects of agent interaction; in this example, agents are employees - people within the company.

Action sequences.

As before, the effects of performing an action sequence are defined in terms of performing, in turn, the elements of the sequence. Again the major difference is that such sequences are no longer anonymously performed but have instead identifiable 'performing' agents. A recursive scheme to this effect is given by axioms thirteen and fourteen above; in effect this defines an agent as having performed a sequence of actions if and only if that agent performs each action in the sequence, one after another, in the order given by the sequence.

At first this may seem a little too strict, especially if we consider that the changes in a system are likely to be performed by a number of agents, performing actions in an interleaved manner and not by a series of action sequences each wholly performed by a specific agent. However, the interleaved view of an agent performing a sequence of actions may still be taken if we consider the frame rule/assumption that we spoke about in chapter two. As an illustration of action sequences we may examine the operators which extend permission and obligation from atomic actions to sequences of such actions, the general definition of which is given by axioms twenty to twenty three inclusive. Take for example the following formula:

$$PS(A, \langle a, b, c \rangle)$$

which by definition is equivalent to

$$P(A, a) \wedge [A, a] P(A, b) \wedge [A, a] [A, b] P(A, c).$$

It is the last of these conjuncts which actually describes the effects of performing one action after another, it may be paraphrased as " the scenario resulting from the agent A performing action a will be one in which the property $[A, b] P(A, c)$ holds. Let us assume then that agent A does in fact perform a and the system is currently in a scenario in which $[A, b] P(A, c)$ holds. Assume further that in this scenario a different agent, B say, performs an action d, then as long as this action does not effect the property $[A, b] P(A, c)$ an appeal to our informal frame rule will still preserve the fact that if agent A performs action b the resulting scenario will be one in which A is permitted to perform action c. Hence modal formulae of the form $[A, b] P(A, c)$ are treated in the same way as non modal formulae such as $P(A, b)$ when it comes to preserving the properties of a scenario not explicitly mentioned by some action.

It is also worth pointing out that we are not totally at the mercy of the informal frame rule; we may for instance decide to explicitly preserve an agent's sequence of actions over all other actions performed by any other agent. Hence

$$[A, a] [A, b] P(A, c)$$

may be replaced by something stronger, such as

$$[A, a] (\forall B \forall \alpha (\neg (A = B) \wedge P(\alpha)) \rightarrow ([B, \alpha] [A, b] P(A, c)))$$

or less drastically by ensuring that specific agents performing specific actions preserve certain modal formulae. However, it is still a lot cleaner to assume that such expressions are preserved until explicitly altered, just as it is in the case of non modal formulae.

The interpretation of obligation sequences described by the operator OS is identical to that of PS except that we will need to augment our model constraints in order to retain the dynamic (transitory) aspects of obligations in this more general context. We shall delay a treatment of this until we discuss how models of DL specifications differ from those of CDAL specifications.

Combinators.

As far as primary action combinators are concerned, we need only observe that actions constructed from such combinators are themselves actions. Hence, when an agent performs such an action it is performing an operationally atomic act which is characterised in terms of some other more primitive actions that that agent can perform. Axioms twenty-four to twenty-nine inclusive are intended to capture this. Note that these combinators combine actions specific to an agent and not agents or agent - action pairs. We shall comment on the possibilities of these latter kind of combinators in the conclusion of this thesis. As a last remark, we ask the reader to note that non atomic actions, whilst taking all or part of their description from the actions of which they are composed, can still be seen as just another action governed by the same rules as any primitive action.

Secondary action combinators have also become less impersonal; instead of operators to say when a particular property is either obligatory or permissible we have operators that define when it is permitted or obligatory that specific agents bring about certain properties. As before the operators 'Per' and 'Obl' are used to describe when an agent is permitted or obliged to bring about a property via a single action whereas 'Sper' and 'Sobl' are used to bring about properties by performing some sequence of actions. A definition of these operators is given by axioms thirty to thirty-three, inclusive; the last two of which define 'Sper' and 'Sobl' in terms of the PS/OS operators we discussed earlier. As an illustration of the use of these operators, consider the following expression, in which Sam, Tom and Mary are agents and the predicate Happy signifies when an agent is happy:

Per(Sam, Per(Mary, Sobl(Tom, Happy(Sam)))).

This says that "there exists some action which Sam is permitted to perform and which will result in a scenario in which 'Per(Mary, Sobl(Tom, Happy(Sam)))' will hold", but this in turn says that "there exists some action which Mary is permitted to perform and which will result in a scenario in which 'Sobl(Tom, Happy(Sam))' will hold". But again

this latter property says that "there exists some sequence of actions which Tom is obliged to perform and which will result in a scenario in which Sam is happy".

Thus the original expression is permitting Sam to perform an action which will permit Mary to oblige Tom to perform some sequence of actions that will result in Sam being happy. As before this does not mention any specific action and the focus of attention is on the property to be achieved and, additionally, which agent is to achieve it.

Models.

The majority of the changes required to our definition of models are due to the interplay between potential agent interaction and the dynamic aspects of obligations. These changes can best be motivated by considering a number of possible transition sequences. For ease of reference we shall repeat the symbolic forms of these conditions above the relevant passages.

Conditions:

- $\forall s \in \mathcal{D} \quad \forall A \in \text{Agt} \quad \forall \alpha \in \text{Act}$
 $\underline{\text{if}} [A, \alpha] \varphi \in \text{TH}(s) \quad \underline{\text{then}} \quad \varphi \in \text{TH}(f_{\langle A, \alpha \rangle}(s))$
- $\forall s \in \mathcal{D} \quad \forall A \in \text{Agt} \quad \forall \alpha \in \text{Act}$
 $\underline{\text{if}} \neg [A, \alpha] \varphi \in \text{TH}(s) \quad \underline{\text{then}} \quad \varphi \notin \text{TH}(f_{\langle A, \alpha \rangle}(s))$

These model constraints have changed very little and have only altered to reflect the fact that actions are no longer anonymous objects. Hence, given that $[A, \alpha] \varphi$ is a property of a scenario s , φ is a property of the scenario reached by applying the function $f_{\langle A, \alpha \rangle}$ to s . Similarly, if $\neg [A, \alpha] \varphi$ is a property of any scenario s , then φ is not a property of the function resulting from applying $f_{\langle A, \alpha \rangle}$ to s . Note that all that has changed from our previous rendering of these constraints is that model level scenario transforming functions are indexed by agents as well as actions. As we have already seen this is to reflect the fact actions are performed by agents.

Condition:

- $\forall s \in \mathcal{D}, \forall A, B \in \text{Agt} \quad \forall \alpha, \beta \in \text{Act}$
 $\underline{\text{if}} [A, \alpha] O(B, \beta) \notin \text{TH}(s) \quad \underline{\text{then}} \quad O(B, \beta) \notin \text{TH}(f_{\langle A, \alpha \rangle}(s))$

This condition on models has also only slightly changed; the reader will remember that its original purpose was to prohibit the default persistency of obligations. The idea was that an obligation had to be explicitly described as the result of an action before it could come about. The above preserves this idea by requiring that for some agent B an action β is obliged only if some agent (perhaps B itself) performs an action which is explicitly described as resulting in B being obliged to do β .

Condition:

- $\forall s \in \mathcal{D}, \forall A, B \in \text{Agt} \quad \forall \alpha, \beta \in \text{Act}$
 if $A \neq B$ and $O(B, \beta) \in \text{TH}(s)$ and $[A, \alpha] \neg O(B, \beta) \notin \text{TH}(s)$
 then $[A, \alpha] O(B, \beta) \in \text{TH}(s)$

This is a completely new constraint and is needed in order to weaken the previous one. The reason why the previous condition on models is too strong is that it forces all agents to explicitly describe whether or not they preserve other agent's outstanding obligations. In order to see this more clearly, consider the following situation: assume that some agent has performed an action which has resulted in a scenario in which the agent B has an obligation to perform the action β , in addition to this assume that in the same scenario there exists an action δ which when performed by the agent D leads to a scenario in which it is permitted for agent A to perform action α . We may represent this scenario more succinctly by the following formulae:

- $O(B, \beta) \text{ and } [D, \delta] P(A, \alpha).$

Suppose now that agent D performed action δ , then according to the previous condition $O(B, \beta)$ would only be in the resulting scenario if it were also the case that $[D, \delta] O(B, \beta)$ is also a property of s . In other words, for all but the obliged agents, all agents would need to explicitly describe whether or not they effected the outstanding set of obligations. In general this would be too cumbersome and result in specifications becoming cluttered and unmanageable. The condition above is intended to remedy this by ensuring that for a specific obligation all agents other than the obliged one, (in the absence of explicit information to the contrary) preserve the obligation when performing their actions. Note that the agent under obligation is excluded from this in order to preserve the reverse default - namely, that when an obligation is discharged by an agent it is not preserved unless of course we explicitly require this. This is needed if we are to retain our original view concerning the transitory nature of obligations.

Conditions:

- $\forall s \in \mathcal{D}, \forall A, B \in \text{Agt}, \forall \alpha, \beta, \delta \in \text{Act}, \forall \sigma \in \text{AS}(\text{Act}),$
if $P(A, \alpha) \in \text{TH}(s)$ and $O(A, \delta) \notin \text{TH}(s)$
and $\text{MOS}(A, \sigma) \in \text{TH}(f_{\langle B, \beta \rangle}(s))$ and $\neg P(A, \alpha) \notin \text{TH}(f_{\sigma} f_{\langle B, \beta \rangle}(s))$
then $P(A, \alpha) \in \text{TH}(s) \leftrightarrow P(A, \alpha) \in \text{TH}(f_{\sigma} f_{\langle B, \beta \rangle}(s)).$
- $\forall s \in \mathcal{D}, \forall A, B \in \text{Agt}, \forall \alpha, \beta, \delta \in \text{Act}, \forall \sigma \in \text{AS}(\text{Act}),$
if $\neg P(A, \alpha) \in \text{TH}(s)$ and $O(A, \delta) \notin \text{TH}(s)$
and $\text{MOS}(A, \sigma) \in \text{TH}(f_{\langle B, \beta \rangle}(s))$ and $P(A, \alpha) \notin \text{TH}(f_{\sigma} f_{\langle B, \beta \rangle}(s))$
then $\neg P(A, \alpha) \in \text{TH}(s) \leftrightarrow \neg P(A, \alpha) \in \text{TH}(f_{\sigma} f_{\langle B, \beta \rangle}(s)).$

These last two conditions describe how p-structures are recovered after a sequence of obligations has been discharged. As before we shall only discuss the first of these as the second differs from the first only in that it deals with prohibited (not permitted) actions whereas the first deals with what happens to permitted actions.

This constraint differs significantly from the model constraints used to recover p-structures after the discharging of anonymous obligation sequences. As before, we may assert the consequent of the conditional, namely that all permissible actions for an agent A are present in a scenario s if and only if they are present in the scenario reached by performing the obligation sequence incurred by performing some action β by an agent B . The reason for this equivalence, rather than simply stating that if an action is permitted for agent A then it is also permitted after the incurred obligation sequence is discharged, is, as before, so that incomplete information about permissions is also preserved. It is time now to see what restriction each of the conjuncts of the antecedent of this conditional places upon this consequent. As before we shall explore each conjunct in turn.

$P(A, \alpha) \in \text{TH}(s)$

This is a very straightforward restriction and quite simply requires that, in order for a permitted action to be reinstated after an obligation sequence, it must have existed before the obligation sequence occurred.

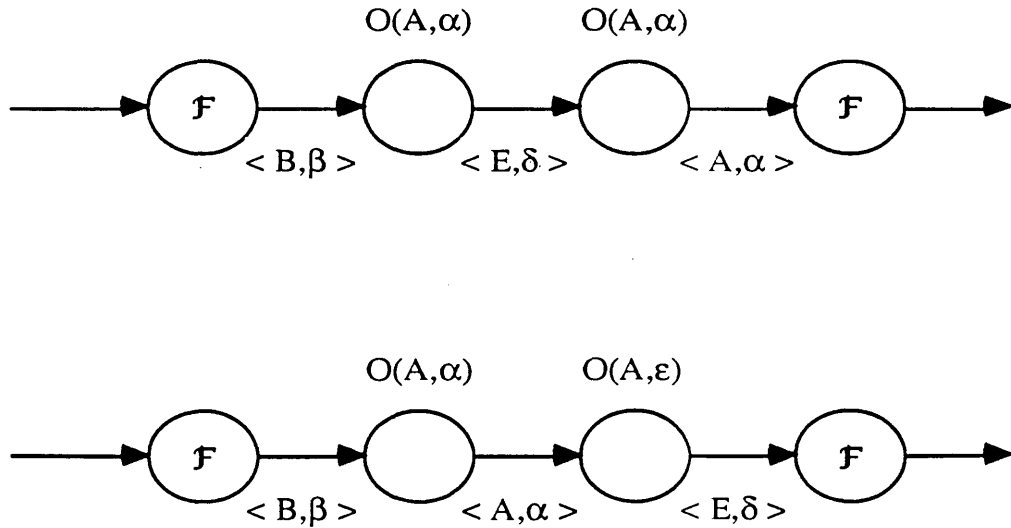
$$O(A, \delta) \notin TH(s)$$

This next condition is needed in order to avoid preserving the p-structures pertaining at the times when an obligatory action for the Agent A exists. As we know, these are only transitory p-structures and so should not be taken into account.

$$MOS(A, \sigma) \in TH(f_{\langle B, \beta \rangle}(s))$$

This says that some agent B performs an action β which results in the agent A incurring a maximal obligation sequence σ . Note that we are using the new (weaker) definition of what it means to incur an obligation sequence. This is because in a multi-agent system it is often the case that either some agents preserve the obligations incurred by other agents or they explicitly change such obligations. In order to model this it is necessary to consider sequences of agent - action pairs in general and maximal obligation sequences of these in particular.

As an illustration of this, consider the following transition cases:



In the above, $\mathbf{F}$ denotes scenarios which are obligation free for the agent A , the arcs are labelled with the name of the transition and the nodes are marked with various properties. The first of these transition diagrams depicts the situation in which an agent distinct from A preserves an obligation on A; the maximal obligation sequence in this

case is $\langle \langle B, \beta \rangle \rangle :: \langle E, \delta \rangle :: \langle A, \alpha \rangle$. This is not hard to see if we consider that the definition of such a sequence (for an agent A) is a series of actions in which all except the last result in the agent A incurring an obligation to perform some action.

The second diagram portrays another example of agent interaction; here, the obligation sequence for A is directly affected by another agent before A has a chance to discharge its last obligation. To be more specific, the agent E overrides A's obligation to perform the action ϵ . In this case the maximal obligation sequence is $\langle \langle B, \beta \rangle \rangle :: \langle A, \alpha \rangle :: \langle E, \delta \rangle$.

Given both these examples of agent interaction it should not be difficult to see why we have been forced to weaken the notion of an obligation sequence to allow sequences of mixed transitions (by many different agents). The idea of maximal sequence is also important since it would be futile to recover a p-structure whilst the agent in question still retained an obligation.

$$\neg P(A, \alpha) \notin TH(f_{\sigma} f_{\langle B, \beta \rangle}(s))$$

This last condition is needed to ensure that after incurring and performing a maximal obligation sequence the last action of this sequence does not alter the fact that before incurring the obligation sequence agent A had permission to perform action α .

If all these conditions are satisfied then the consequent also holds and the actions permissible for agent A (before it incurred a series of obligations) are reinstated.

All things considered, the "frame rules" we gave earlier have changed considerably with the introduction of agents. The primary reason for this change is due to the ability of agents to grant and revoke permissions and obligations. The above discussion should give the reader some feeling for the complexities of interaction (especially concerning the dynamic aspects of obligations) which arise from the introduction of agents. In the next section we intend to show how these features can be exploited in the scope of a more concrete example.

5.4 An Example.

The primary aim of this section is to present and discuss an example of a specification written in DL. The domain we have chosen for this exercise is that of a *simple* telephone exchange system. We stress the use of the word simple because it is not our intention to model all the aspects of such a system; instead we shall focus our attention on how a (telephone) user uses a telephone in order to speak to another user. We chose this domain for mainly two reasons. Firstly, it is a familiar "everyday" system of which almost all readers will have had some experience and, hence, will be able to make their own judgements as to how well DL captures its structural and behavioural properties. The other reason for choosing to specify a telephone system is that it provides a small but sufficiently rich example of various kinds of interaction between the users of the system. This means that the points we wish to make can be made concisely without the need to describe a large infrastructure within which our example sits.

In keeping with the style of previous examples the remainder of this section is structured as follows: first we shall present our example, which will then be followed by a discussion of what the specification captures, pointing out how it may be extended, and finally we shall end by commenting on the shortcomings of DL in the scope of this specific example - we shall delay saying how these may be overcome until the next chapter where we will tackle such shortcomings at a more general level.

A Telephone System.

Language.

Sorts: Telephones, Act, Agt.

Constants: Ex of sort Agt.

Variables of sort:

Telephones: t, t', t'',

Agt: u, u', u''

Predicates:

Rec-up of type < Telephones >

Busy of type < Telephones >

BusyTone *of type* < Telephones >
 RingTone *of type* < Telephones >
 BellRinging *of type* < Telephones >
 Connected *of type* < Telephones, Telephones >

Actions:

Lift *of type* < Telephones >
 Putdown *of type* < Telephones >
 Call *of type* < Telephones, Telephones >
 Connect *of type* < Telephones, Telephones >
 BusyInd *of type* < Telephones >
 RingInd *of type* < Telephones >
 RingBell *of type* < Telephones >

Axioms.

- i $\text{Rec-up}(t) \rightarrow \text{Busy}(t)$
- ii $(\neg \text{Busy}(t) \vee \neg \text{Busy}(t')) \rightarrow \neg \text{Connected}(t, t')$
- iii $[u, \text{Lift}(t)] \text{Rec-up}(t)$
- iv $[u, \text{Putdown}(t)] \neg \text{Rec-up}(t)$
- v $[\text{Ex}, \text{BusyInd}(t)] \text{BusyTone}(t)$
- vi $[\text{Ex}, \text{RingInd}(t)] \text{RingTone}(t)$
- vii $[\text{Ex}, \text{RingBell}(t)] \text{BellRinging}(t)$
- viii $[\text{Ex}, \text{RingInd}(t) \parallel \text{RingBell}(t')] [u, \text{Lift}(t')] \text{Connected}(t, t')$
- ix $\text{Connected}(t, t')$
 $\rightarrow [u, \text{Putdown}(t)]$
 $(\neg \text{Busy}(t) \wedge [u', \text{Putdown}(t')] \neg \text{Busy}(t'))$

- x $\text{Connected}(t, t')$
 $\rightarrow [u, \text{Putdown}(t')]$
 $[u', \text{Putdown}(t)] \neg \text{Busy}(t) \wedge \neg \text{Busy}(t')$
- xi $\neg P(u, \text{Call}(t, t))$
- xii $\text{Rec-up}(t) \leftrightarrow P(u, \text{Putdown}(t))$
- xiii $\neg \text{Rec-up}(t) \leftrightarrow P(u, \text{Lift}(t))$
- xiv $[u, \text{Call}(t, t')] O(\text{Ex}, \text{Connect}(t, t'))$
- xv $\text{Busy}(t') \rightarrow [\text{Ex}, \text{Connect}(t, t')] O(\text{Ex}, \text{BusyInd}(t))$
- xvi $\neg \text{Busy}(t') \rightarrow [\text{Ex}, \text{Connect}(t, t')] O(\text{Ex}, \text{RingInd}(t) \parallel \text{RingBell}(t'))$
- xvii $O(\text{Ex}, \text{Connect}(t, t'))$
 $\rightarrow [u, \text{Putdown}(t)]$
 $\neg O(\text{Ex}, \text{Connect}(t, t')) \wedge \neg \text{Busy}(t)$
- xviii $O(\text{Ex}, \text{RingInd}(t) \parallel \text{RingBell}(t'))$
 $\rightarrow [u, \text{Putdown}(t)]$
 $\neg O(\text{Ex}, \text{RingInd}(t) \parallel \text{RingBell}(t')) \wedge \neg \text{Busy}(t)$
- xix $O(\text{Ex}, \text{RingInd}(t) \parallel \text{RingBell}(t'))$
 $\rightarrow [u, \text{Lift}(t')]$
 $\neg O(\text{Ex}, \text{RingInd}(t) \parallel \text{RingBell}(t'))$

As usual we shall look at the language component first; this is comprised of three sorts of entities, the logical categories of actions and agents and the extra logical sort Telephones. The agent sort should be interpreted as the domain of telephone users of which Ex (signifying the telephone exchange) is a special element. The Telephone sort should be interpreted in the obvious way and is populated by the telephones in the telephone system. The action sort has as elements the following actions.

'Lift' is intended to represent the lifting of a telephone receiver and hence has as parameter the specific telephone whose receiver is to be lifted.

'Putdown' is the name of the action corresponding to placing the receiver back onto the base of a telephone and is therefore also parameterised with a specific telephone.

'Call' is the name of the action a user performs when (s)he dials one telephone from another; the action is parameterised by two telephones, the first of which signifies the telephone (s)he is calling from and the second the telephone which (s)he is calling.

'Connect' is the name of the action which the exchange performs when trying to establish a connection between two telephones. Its parameters are the telephones between which the connection is to be established.

'BusyInd' is the name of the action the exchange performs when it wishes to send a busy tone to a telephone whose user is trying to call a phone which is already engaged. This action is parameterised by the telephone to which the busy tone is to be sent.

'RingInd' is the name of the action the exchange performs when it wishes to send a ring tone to a telephone to signify to the user of that telephone that the telephone he wishes to be connected with is in fact ringing. As before the action is parameterised with respect to the telephone to which the ring tone should be sent.

'RingBell' is the name of the action the exchange performs when it wishes to start the bell of a specific telephone ringing; again the action is parameterised with respect to the telephone to which this should be done.

In addition to these three sorts there are a number of predicates, all but one of which are used to record specific properties of telephones. The 'Rec-up' predicate is used to signify when the receiver of a telephone is off the hook; note that the negation of this predicate is intended to denote the property that a telephone's receiver is placed properly on the telephone base. The 'Busy' predicate is intended to denote when a telephone is busy, the negation of this property signifies that a telephone is free to accept calls. The remaining monadic predicates, namely 'BusyTone', 'RingTone' and 'BellRinging', signify that a specific telephone is receiving an engaged signal, receiving a ring tone (which means the phone called using that phone is ringing) and the telephone is ringing (which means someone is trying to call it), respectively.

Finally, the predicate 'Connected' describes an ordered relationship between two telephones. A telephone *t* is connected to a telephone *t'* if and only if a connection has

been established between a user calling from telephone t and a user receiving this call at telephone t' . The order is significant because the first parameter is used for the telephone from which a call is made, whereas the second is used to denote the telephone to which the call is being made (and this is significant in terms of what may happen when one of them hangs up).

The description of the telephone system (both structural and behavioural) is given by axioms i to xix. Rather than adopt our previous (rather linear) style of describing and discussing each axiom in turn we shall instead discuss general observations of everyday telephone usage, referring to the appropriate axioms as necessary. Apart from giving the axioms a certain context, this will also be helpful in judging how well the specification models the real world structure and usage of an operational telephone system.

Perhaps the first activities that a user of a telephone must negotiate are the acts of lifting and replacing a telephone receiver. Assuming that the user is free to act without interference from any other source there are two conditions governing the acts of lifting and replacing telephone receivers. The first is that a telephone receiver may not be lifted at a time when the receiver is already displaced and the second is that it may not be replaced when it has not been first removed from the telephone body.

Although these may seem like trivial and obvious observations they do bring to light some interesting questions about the 'correct' way to represent them. Our approach is reflected in axioms xii and xiii. The first of these says that a user is permitted to replace the receiver of a telephone (by performing the 'Putdown' action) if and only if the receiver of that telephone is displaced (indicated by the predicate 'Rec-up'). The second is similar and permits a user to lift a telephone receiver (by performing the 'Lift' action) if and only if the receiver is not displaced (indicated by the negation of the predicate 'Rec-up').

Thus, in addition to saying when a receiver is prohibited either from being lifted or put down, xii and xiii also prescribe when a receiver *is* permitted to be lifted or put down. A further consequence of these axioms is that a telephone receiver may not (in any single scenario) be both permitted to be lifted and put down. Equivalently, we may say that it is always the case that a user may either lift or put down a telephone's receiver.

We can now build up a fuller picture by inspecting the descriptions of the 'Lift' and 'Putdown' actions. These are given by axioms iii and iv, respectively. Imagine that the

receiver of a particular telephone t is placed firmly on the telephone; then by axiom xiii we know that a user is permitted to lift that receiver. Further, suppose that he proceeds to do so; then axiom iii says that as a result of his/her action the receiver is then deemed to be displaced or 'up'. This in turn, by virtue of axiom xii, means that the user is no longer permitted to lift the receiver but is permitted to put it down. Suppose then, that the telephone is put down; a consequence of this, given by axiom iv, would be that the receiver would be deemed 'down'. Again, this in turn will, as a consequence of axiom xiii, result in a scenario where the receiver may be lifted but is prohibited from being put down. Hence, the interaction between the description and prescription of the 'Lift' and 'Putdown' actions leads to an alternating cycle between when the user of a telephone may or may not lift or put down that telephone's receiver.

This is by no means the only way in which we could have modelled the usage of telephone receivers. One may, for instance, take the view that prohibiting a receiver from being lifted when it is already displaced or 'up' is a physical impossibility rather than a normative condition. If one were to take this view (which is not to our minds unreasonable) one could place more emphasis on the descriptions of actions. One way in which this could be achieved is by restricting the descriptions of the 'Lift' and 'Putdown' actions to the relevant scenarios, remaining silent about the result of these actions in other scenarios. Consider for example the following expressions:

$$\neg \text{Rec-up}(t) \rightarrow [u, \text{Lift}(t)] \text{Rec-up}(t)$$

$$\text{Rec-up}(t) \rightarrow [u, \text{Putdown}(t)] \neg \text{Rec-up}(t).$$

The first of these describes the result of lifting a telephone receiver in scenarios where the receiver is not already displaced, whereas the second describes the result of putting a receiver down in scenarios where it is displaced. Neither expression comments on the anomalous situations in which a receiver is lifted/put down when the receiver is already deemed to be 'up' or 'down' respectively. Note that, in any refinement and ultimately any implementation of this specification, these anomalies cannot be realised even if we so wished. In a sense, it is as though we are borne out by reality when we remain silent about such impossibilities. Given this view we may be less restrictive in our prescription of the 'Lift' and 'Putdown' actions. We could for example state that both actions are always (unconditionally) permitted since it would be impossible to implement the anomalous cases. Alternatively, we could be more guarded and retain our previous formulation; this latter combination would not only highlight the physical impossibility of the anomalies we have mentioned but also prescribe their omission.

Throughout our discussion of lifting and replacing a telephone's receiver the property of whether or not the receiver is displaced or not has been central. The predicate used for this purpose is 'Rec-up'; 'Rec-up' also plays another role in our system, it is used to indicate one situation in which a telephone is considered to be busy. This is reflected in axiom i, which tells us that in all scenarios in which a telephone's receiver is 'up' that telephone is also considered to be busy. The busy status of a telephone is modelled by the 'Busy' predicate which signifies when a phone is not free to receive a call. Note that as things stand, this definition will admit scenarios in which a telephone is busy without its receiver being displaced. One such situation may arise when a telephone is not connected to an exchange and therefore has no connection to the telephone network. In this case the receiver may not be displaced yet the telephone is unable to accept any call. It should also be noted that axiom i says nothing about the busy status of a telephone when that telephone's receiver is not 'up'.

Given this connection between a telephone's receiver being 'up' and it being busy, it is a simple matter to see that the lift action description given in axiom iii means that lifting a telephone receiver implicitly causes that telephone to become busy. Observe that the reverse action of replacing the receiver does not mean that the telephone becomes free. Later when we see the various contexts in which receivers are replaced we shall see the reason for this.

Having negotiated the acts concerned with managing the telephone receiver, a purposeful user would then wish to make his/her telephone call. In reality this would in itself involve a number of simpler actions such as waiting for a dial tone and then dialing a number of digits. In order to be compact we have simplified these sequence of operations into a single 'Call' action. The intuition behind this action has already been given; its more formal description can be found in axiom xiv. In essence this action has but one effect and that is to impose an obligation on the telephone exchange to connect the telephone from where the user is calling to the telephone he has called. We can look upon this as a level of indirection, where the telephone exchange mediates between users of the telephone system by providing a service to deal with requests from users to link up specific telephones.

Hence, axiom xiv can be regarded as describing the 'Call' action as a request to the telephone exchange for a connection from telephone t to telephone t' . Where t is the telephone from where the request is being made and t' is the telephone to which connection is desired.

We have purposely omitted a prescription of when the 'Call' action is permitted; there are several reasons for this, the primary one concerning our omission of a more detailed telephone dialling procedure. Had we for instance included an action of waiting for a dial tone it would have been simple to identify the proper place to prescribe the 'Call' action. Another reason for not explicitly prescribing this action is that our current simpleminded characterisation does not distinguish between those telephones that can be called and those which cannot. It would be ridiculous, for instance, to say that a user is permitted to call a telephone not yet connected to the telephone system. In a more elaborate specification we might rectify this by introducing subscriptions and valid telephone numbers.

On the other hand if we assume that all telephones are implicitly 'connectable', then we may modify axioms iii and xiv as follows:

iii' $[u, \text{Lift}(t)] \ (\text{Rec-up}(t) \wedge (\neg (t = t') \rightarrow P(u', \text{Call}(t, t'))))$

xiv' $[u, \text{Call}(t, t')] \ (\text{O}(\text{Ex}, \text{Connect}(t, t')) \wedge (\neg P(u', \text{Call}(t, t'))))$.

Here, axiom iii has been changed so that every time a receiver is lifted not only is the receiver deemed 'up' but permission is also given to any user u' to call any telephone t' as long as t' is distinct from the telephone being used to make the call. Note also that we have used u' to indicate that the user making the call may be different from the user who lifted the receiver. The permission given by iii is then revoked by a user utilising that permission and so making a call. This can be seen in the modified axiom xiv above, where permission to make any further calls (from any user) from the telephone in question are not permitted. This cycle can then be reiterated by the telephone's receiver being replaced and picked up anew.

The condition in iii', which guards against permitting a user from calling the same telephone he is using to make the call can best be explained by axiom xi. This axiom prohibits a user from trying to call the same telephone he is calling from. This would seem like a reasonable constraint given that such calls - on the whole - would be non productive.

Leading on from the fact that the effect of performing a 'Call' action is to oblige the telephone exchange to connect the appropriate telephones, we shall now examine how this may be achieved. The description of the 'Connect' action is given in axioms xv and xvi. Axiom xv describes the effects of performing the 'Connect' action when the telephone being called is busy, whereas axiom xvi describes its effect when the

telephone being called is not busy.

In the former case the exchange incurs an obligation to send a busy indication signal to the telephone being used to establish the call. The action name is 'BusyInd' and its effect, as found in axiom v, is to establish a busy tone in the receiver of a telephone. This is denoted by the predicate 'Busytone'.

In the case where the exchange performs the 'Connect' action in a scenario in which the telephone being called is not busy the exchange incurs an obligation to perform the parallel action of notifying the caller of a ring tone and ringing the bell of the called telephone. These separate actions are described in axioms vi and vii respectively and should be interpreted in a similar manner to the 'BusyInd' action. Note, however, that the telephone exchange incurs an obligation to perform the parallel composition of these actions. The (minimal) effect of which is the conjunction of the effects of both these individual actions. This seems like a rather good use of the parallel composition combinator as it does not force either a truly concurrent implementation of these actions nor a specific serialisation.

The preceding description of the 'Connect' action gives an account of what happens when an exchange discharges an obligation to connect two telephones. Things may, however, not proceed this far. A user may terminate a request to the exchange by replacing his receiver before the exchange has an opportunity to discharge its obligation. In other words, it is conceivable that a user can make a call and replace the receiver of the telephone he is using to make the call before the telephone exchange gives him an indication of whether the telephone he is calling is busy or not. This situation is handled by axiom xvii. This describes the action of putting down the receiver of a telephone *t* in a scenario in which the telephone exchange still has an obligation to connect *t* to some telephone *t'*. The effect is that the outstanding obligation is cancelled and the telephone whose receiver is replaced is considered to be free. Note that there is no need to cancel such an obligation in the case where the exchange performs the 'Connect' action as performing an obliged act carries with it its own intrinsic cancellation.

Returning now to our persistent caller and assuming that he has reached the stage where the telephone exchange has an obligation to send a ring bell signal to the telephone he is calling to and a ring tone signal to the telephone receiver he is using to call from, one of three things may happen. We shall examine each of these in turn.

One possibility is that the exchange fulfills its obligation and sets the called telephone ringing and indicates a ring tone to the caller via the calling telephone. We have already observed that the basic effect of this action will be the combined effects of the two distinct actions 'RingInd' and 'Ringbell'. Axiom viii goes one step further and describes what would happen if a user lifts the receiver of the telephone which is now ringing as a result of the exchange trying to establish a connection. The result is of course that the telephones in question are considered connected (and conversation between the users of the two telephones may take place). Note the use of an iterated modality in the description of this property; the occurrence of the parallel action acts as a context indicator for the occurrence of the 'Lift' receiver action.

Before we examine how to terminate a conversation, we shall examine the other two possibilities. Both of these alternatives are concerned with the potential occurrence of actions which may discharge the telephone exchange's obligation to perform the parallel action of a 'RingInd' and 'RingBell' composition before the exchange has a chance to fulfill this obligation in the natural way (by performing the obliged act).

One way in which this can happen is if the user who has requested the call puts down the receiver before the exchange has had a chance to carry out the obliged act. The effect of this action is described in axiom xviii and amounts to cancelling the obligation on the exchange to perform the parallel ring composition and freeing the telephone whose receiver has been replaced.

The other way involves some user lifting the receiver of the telephone which is to be called, again before the telephone exchange has had a chance to perform the parallel ring composition. This situation is partially handled by axiom xix. An obvious effect of such an action is that the obligation on the telephone exchange to perform the parallel ring composition is revoked. But this is only part of the story as this does not tell us what happens to the caller - does he receive a busy tone, for instance, or is he simply kept in the dark, his only option to replace the receiver of his telephone ? The latter is already accommodated by axiom xix whereas the former may be incorporated by modifying axiom xix as follows:

$$\begin{aligned}
 \text{xix'} \quad & O(Ex, RingInd(t) \parallel RingBell(t')) \\
 & \rightarrow [u, Lift(t')] \\
 & \neg O(Ex, RingInd(t) \parallel RingBell(t')) \wedge O(Ex, BusyInd(t)).
 \end{aligned}$$

Assuming that it is the first of these three possibilities that materialises and a connection has been established and a conversation is in progress, the last operation we need to describe is the termination of a call. This aspect is characterised by axioms ix and x. Axiom ix describes the action of putting down the receiver of the telephone from which a connected call was initiated. The effect is that the telephone whose receiver has been replaced is no longer considered busy and *if* the receiver of the called telephone is then also replaced, it is also considered to be free. Note that because of axiom ii, as soon as the calling telephone's receiver is put down the connection between the two telephones is severed.

Axiom x on the other hand describes the situation in which the receiver of the telephone which is the recipient of a connection request is put down. In this situation the effect of the action is not to sever the connection but to conditionalise this to the further act of the initiating user putting down his receiver. Once this has been done both phones are considered to be free and hence by axiom ii the connection is also considered to be severed.

The differences between axioms ix and x stem essentially from whether the initiator or recipient of a call takes the lead in terminating a call. The reader will have observed (whilst using a real telephone system) that only the initiator of a call has the privilege of termination. If the recipient tries to initiate a termination by replacing his/her receiver he/she will often find that if the initiator of the call has not replaced his/her receiver the call is still connected when he/she tries to make another call.

Although this example does not illustrate, in an agent context, the use of secondary action combinators, nor the use of sequential or nondeterministic primary action combinators, these may be accommodated by extending the role of the telephone exchange. One way in which this could be achieved is by the introduction of a network of telephone lines - the exchange being made responsible for the routing of telephone calls. Such an extension would not be dissimilar to the network example of the previous chapter, requiring each telephone to be associated with a local exchange. Local calls could then be handled by the local exchange whereas non local calls would need to be communicated to other exchanges. It is not hard to see how such an extension could evolve into quite a large specification.

Perhaps the major benefit of introducing agents has been one of structuring; in terms of both the structural and behavioural properties of the system. The agent indexical allows

one to consider specific components of a system in isolation, enabling one to focus on that particular component's structural and behavioural properties. Once this has been achieved it is then also possible to consider how such components relate to each other.

Another benefit of agent indexicals is that they may help to indicate possible implementation strategies. It is possible, for instance, that some components of a system have a stronger relationship, in terms of interaction, than others. Agent identification might help to highlight this at an early stage and hence lead to a closer implementation relationship - sharing computing resources, for example. Of course a conceptual relationship need not necessarily manifest itself in such a direct manner; it would be a mistake therefore to assume this *per se*.

The idea of structuring through the use of agents can be taken much further than we have done here. A particularly useful extension would be the inclusion of an explicit agent typing structure. As things stand we admit only one monolithic type whose elements are attributed with the ability to perform actions. It is often the case that we may wish to discriminate between different types of agents.

Our telephone example provides us with a good example of this because telephone users and telephone exchanges are different kinds of agents. Although it was convenient in this instance to consider the 'exchange' as a particular telephone user it may cause confusion if we were to introduce more 'exchanges'. Problems may begin to arise with the operational roles of agents, for example can an exchange initiate a call ? Quantification and individuation can also become unclear - what is the difference between an agent type and a specific instance of that type ? Again, although such issues may be handled by relativisation predicates, it may be more appropriate - in the interests of clarity, if nought else - to incorporate an explicit typing structure. We believe that the proper place for this should be at the time of creating a specification language; the form and machinery that such a structure would need would of course depend upon the design of such a language.

Further structuring may be achieved if we consider more operational relationships between agents. So far we have provided a fairly selfish view of the world. In our scheme of things agents perform actions without the help of other agents. Fortunately the real world is not so segmentary; agents co-operate with each other to perform certain actions and regulate themselves (in accordance with plans) in order to achieve certain goals.

Both these aspects of agent interaction can be handled with suitable extensions to our present notion of action combinators. We have not yet fixed on a formal treatment of these extensions and will therefore only provide an informal sketch of what may be done. What we have in mind can be seen by the following expressions:

- i $[A \oplus B, \alpha] \varphi$
- ii $[A, \alpha \oplus B, \beta] \varphi$

Here $\oplus$ denotes some infix combinatory functor such as 'in parallel' or 'nondeterministically'. The first expression is to be read as "some combination of agents A and B performing the action α results in the property φ ". One option is to consider the combination of agents $A \oplus B$ as another agent; so just as primary action combinators take actions and form more complex actions, agent combinators take agents and produce more complex agents. Although this looks like a promising approach to modelling things like co-operation between agents we are not yet in a position to go much further.

The second expression is perhaps more conventional; CCS [Mi80] and CSP [Ho83] are both languages in which such combinators are used. Substituting the term agent for process both these formalisms employ combinators which form systems in which distinct agents may perform distinct actions in some combination. The analogy is far from perfect but does illustrate the intent of such combinators. Again, we are not in a position to say more on this topic except to point out that such extensions may be more at home in a framework capable of expressing and reasoning about the concurrent execution of actions. We shall return to this point in the next chapter after we have commented on how our present framework may be extended to handle concurrent execution.

As a final remark to this chapter we must point out that the reasoning (deductive) aspect of our logic remains unchanged by the introduction of agent indexicals. However, the definition of models for specifications written in this extended formalism has changed considerably in order to cope with the explicit derogation of obligations. In the next chapter we shall summarise our achievements, identify our shortcomings and point to future work.

CHAPTER SIX

CONCLUSIONS

6.1 Introduction.

Although each chapter of this dissertation concludes with brief summaries and remarks concerning the achievements and shortcomings of each chapter, we shall end our work by outlining (at a more general level) how the framework we have developed can be exploited and extended. We shall start by reiterating the major contributions of our work.

Throughout our exposition we have been at pains to stress the importance of separating the prescriptive and descriptive aspects of a system's dynamics. This separation of concerns and all that it brings - benefits and problems alike - is the major force of our work. The main motivation for this stemmed from a confusion concerning the use of pre-conditions in the pre/post-condition tradition of action description and introducing deontic operators to cope with action prescription helps to achieve a cleaner (less confused) approach to specifying the behaviour of a system. This, as we saw earlier, is especially true in the case of the description of causality between agents.

This separation of concerns has many advantages over the more conventional pre/post-condition method of describing actions. In the latter the description of a system's behaviour is implicitly assumed to be present in the definition of the action. In other words, pre-conditions are used as a means of both defining the effects of an action and when it may be performed. In contrast to this, we have put forward a framework in which the idea of a pre-condition is used to *define* the context in which an action is performed and where an explicit deontic operator is used to represent *when* an action may be performed. This shift to an explicit characterisation of behaviour not only simplifies the description of actions - by, in general, reducing the complexity of pre-conditions - but also helps to clearly express the behavioural requirements of a system. This latter is something which is sorely lacking in the implicit pre/post-condition approaches. In order to see this we need only observe the following natural language descriptions of opening a particular window.

- i Whenever the window is closed, opening it results in the window being open.
- ii Whenever it is a fine day it is permitted to open the window.

The first of these, that is i, can easily be accommodated in the conventional pre/post-condition manner of action description. The pre-condition would correspond to the window being closed, the action to opening the window and finally the

post-condition would denote the fact that the window was open. The second sentence, that is ii, has a less natural representation in this approach. Probably the best coding would involve making the property of "being a fine day" a pre-condition, retain the action of opening the window and treat the post-condition as denoting the property that the window is open. However, this does not truly capture the sentence in ii, which says that on "fine days" the window is permitted to be opened. Note that it does not say that on "fine days" if the window is opened the result will be an open window. In other words ii says nothing about the effect of performing the action of opening the window, just the fact that this action is allowed to be performed on "fine days".

An obvious way to strengthen the coding of these sentences would be to treat them uniformly and combine them so that we obtain the following composite sentence.

- iii Whenever the window is closed and it is a fine day the result of opening the window is that the window becomes open.

This is then a little more amenable to the conventional pre/post-condition approach; we may assign the conjunction of the properties "closed window" and "fine day" to the pre-condition, retain the opening window action as the action and have as post-condition the property that the window is open. However, this is far from a satisfactory solution since it forces together the definition of an action in terms of its effects and the use of that action as may be behaviourally desirable.

To see why these two aspects of action description should be kept separate, consider what happens when we wish to augment the description of this action with the fact that "a closed window is not permitted to be opened on rainy days". It is easy to see that this would further increase the complexity of the pre-condition. The problem is that these approaches have no formal correspondence to the deontic notion of being *permitted* to perform an action. The main motivation of our work has been to incorporate this idea of being allowed to perform an action alongside the more conventional description of an action's effects, if performed.

Given this idea of permitting an action, the two sentences, i and ii, may be represented more naturally. The first in the manner already discussed, employing the pre/post-condition style of describing an action's effects, and the second by use of the deontic idea of permitting the open window action (appropriately restricted to "fine days", using "fine day" as a relativisation property).

Having established the need for and the characterisation of this deontic concept of permission, we exploited it to talk about related notions such as refraining from actions, being prohibited (not permitted) to perform an action and incurring obligations to perform actions (restricted to perform only the obliged action). This together with the ability to refer to agents as the performers of actions gives us an expressive framework within which to state the requirement specifications of multi-component systems.

The telephone system example of chapter five shows this very well. As well as characterising the interaction between telephone users and the telephone system's exchange, it also demonstrates the use of the other deontic concepts. The idea that an exchange is obliged to (must) respond to a telephone user's actions or some telephone user is forbidden to perform some action is just as important a part of the requirements of the system as whether or not some telephone user (or telephone exchange for that matter) is permitted to perform actions. Hence, the introduction of an explicit permission concept distinct from the implicit guards of a pre-condition also allows us to express those situations in which an agent is prohibited or obliged to perform a certain action.

Actions which are to be prohibited are considered to be not permitted and those which are obligatory are defined to be the only permitted action available to an agent. The former is denoted by negating (using the classical logic connective $\neg$) the expression denoting a permissible action. The latter is slightly more complicated and involves model level constraints which take the form of deontic frame axioms similar to the more traditional frame axioms [Sh86].

Briefly, the idea is to characterise obligatory actions as (by default) non persistent phenomenon. Hence, incurring an obligation results in a situation in which all "normal" deontic information regarding which actions are permitted or prohibited is suspended, permitting only the obliged action to be performed. When this obligation is discharged, either by the agent incurring the obligation performing the action or another agent revoking the obligation, the deontic information which obtained before the obligation was incurred is reinstated (for the agent involved). However, this assumes that the effect of the action is not to change the more persistent deontic information or does not result in a further obligation. The reader is referred to chapter three to see how these situations are accommodated. The important point we wish to make here is that the introduction of a simple concept of permission has enabled us to introduce a richer collection of behavioural concepts which can be used to express a system's requirements.

Although the model level constraints used to capture this transitory aspect of obligations are not straightforward, we feel that the effort was warranted by the presence of this phenomenon in a wide variety of application areas. The simple stroke of a keyboard signalling an interrupt to a computer is as good an example as any. The machine in question may be permitted to do a number of other housekeeping activities when not servicing the keyboard, these are suspended by the arrival of a keyboard interrupt until the interrupt has been serviced. In such a situation we may say that the machine is obliged to service an interrupt on receipt of the same. If this example is too simplistic, we only have to consider any number of "real world" multi-agent systems with interaction (like the telephone system of chapter five) in which the action performed by one agent results in some other agent incurring an obligation to do some other action.

Another example is the hierarchical "menu" driven interfaces used in some applications; one function of these is to temporarily hide from the user of such applications choices which are not allowed, given previous decisions. Thus at each point, the user is obliged (forced) to choose from a restricted set of options all the time knowing that other choices although allowed in other contexts are temporarily prohibited to him. Although it is not possible to oblige a user to choose between a number of actions in our present framework we hope that future extensions (such as those outlined in chapter three) will allow this. We shall return to this point below.

In summary our work has been directed at harmonising the interface between describing the effects of actions and prescribing the use of such actions in order to characterise the behaviour of a system. The remainder of this chapter will be split between looking at the current shortcomings of our framework (and outlining potential extensions aimed at overcoming them) and a look at how the present framework has been exploited to date.

6.2 *Potential extensions.*

At the outset of this work we stated that it was not our purpose to design a specification language, but to lay the foundational concepts for such a language. Although DL has been used to capture the requirements of various systems it needs further refinement if it is to be successful as such a medium. There are many ways in which this may be achieved.

The inclusion of more standard data structures and associated operators as mentioned in chapter two would be useful to users of such a specification language. Most specification languages have such constructs, usually known as built in data types. These are usually based around application independent structures such as lists, sets, number systems etc which can be used to specify application dependent structures. This should not present too many difficulties given that DL has a classical (sorted) first order logic basis. The rigorous treatment of various data structures given in [Jo86], developed in the VDM specification language provides a good source for such structures.

Quite apart from such conveniences as "built in data types" we also need to look at the more difficult issues of structuring specifications. We have already mentioned the top down mechanism of refining specifications so that a specification may be arrived at through a number of stages, where each intermediary stage suppresses some details and focuses on those aspects pertinent to that stage. This activity has the effect of "building" the specification in a number of phases and so helps in the construction of large specifications. As we mentioned in the introduction of this dissertation, we envisage this refinement process to be supported by the mechanism of logical implementation as described in [MVS85].

In complement to this form of *external* structuring we also need some notion of *internal* structuring. The analogy with modules in programming languages can be used here. One of the ideas behind a programming module is to lend some structure to a program; it divides the program up into a number of manageable units. Similar structuring mechanisms have been proposed for specification languages. Starting with the pioneering work of Liskov and Zilles [LZ75], much has been done in this field especially in the algebraic school of specification, witness the work in [BG80], [Wir83a] and [Go82]. Most of this work is to do with combining small specifications in order to form larger ones. Various combinators have been suggested ranging from a simple

syntactic union of two specifications to more semantically based ones such as parametrising one specification by another. The interested reader is referred to the above references for a more detailed account.

We have not yet decided upon a clear policy for structuring specifications written in DL but hope to benefit from the work done in the algebraic setting and look more closely at the specification (schema) combinators found in the Z specification language [Mo84]. Although the latter is more appropriate for DL, due to the fact that Z is a state transition based formalism, we feel more investigation is needed into the "appropriate" structure of specifications. It is not yet clear to us what, in general, this should be. One possible route to investigate is the use of the translation between theories mechanism which supports logical implementation. We have for instance used this to define the definition of a database view, which is then used to create large database specifications by combining smaller database specifications which are considered to be "views" of the former. The interested reader is referred to [KMS85].

Last, but not least, a specification language needs to have good syntactic structure often referred to as syntactic sugaring because it is often this aspect of specification languages which makes them more pleasant to use (swallow). Again we are sure there are improvements to be made in this area especially when it comes to lengthy expressions which are more easily read when better syntactically presented. This is often a difficult area as it is inherently subjective. Because of this, we have not chosen to adopt a particular style but it is hoped that as the use of DL grows, feedback from "users" will be used to improve the syntax. In all probability, different groups, perhaps tackling different families of systems, will adopt their own style.

So far we have mentioned improvements which could be made if we wished to extend DL into a usable specification language; we shall now turn our attention to some of the shortcomings (or lack) of the concepts within DL. Most of those which we mention below have been addressed earlier in the appropriate chapter and we reiterate them here by way of a summary. There are three major areas for improvement: the first concerns the weakening of the concept of obligation, the second involves the deontic treatment of non-primitive actions and the third identifies the need for some way in which to incorporate time in our framework to enable us to express temporal properties. We shall now look at each of these in turn.

We feel that the case for a weaker notion of obligations has already been made quite strongly. However, in order to reinforce this we need only look at the natural language usage of the word. When a person incurs an obligation to do something, he does not usually lose permission to do everything else. A more natural assumption is to think of him losing permission to do those actions which are at odds with his obligation. The extent of what he is prohibited to do whilst under an obligation depends very much on the nature of the obligation and its priority compared to other actions the person may choose to do. This can perhaps be better seen if we substitute the word *should* for *obligation*. Just because a person should do something does not by itself prohibit them from doing other things.

In chapter three we suggested that one possible *solution* would be to oblige, not just a singleton action, but a whole set of actions. This would have the interpretation that one action within the set specified must be the next action performed. Another way to weaken the force of obligatory actions is to augment the present obligation of a specific action with a specific set of actions which are permitted whilst that obligation is not discharged. This may for instance take the form " $O(a, \{\alpha \mid A\})$ " where 'a' denotes the agent, ' α ' denotes the action to be obliged and 'A' denotes the set of actions which are tolerated whilst the obligation upon agent 'a' to perform action ' α ' persists. This operator can easily be implemented using the machinery of obliging sets of actions mentioned in chapter three.

One consequence of weakening our present obligation operator to operate on sets of actions is that it will allow us to prioritise the importance of actions. We could, for instance, construct a lattice of tolerance sets (sets of actions allowed whilst an obligation is present) assigning smaller sets to those actions which are important and large ones to those which are not. We are currently exploring this area.

Another area which needs further investigation is the distribution of the deontic operators over non-primitive actions. At present the combinators we have explored form actions which can be treated as actions separate from their constituent actions. Hence, the prescription of such actions need not be concerned with the prescription of their constituents. This approach abstracts from the operational flavour of the combinators. Thus instead of saying we are permitted to do *either* the action α or the action β , we say we are permitted to do *the* action $\alpha+\beta$. In contrast it would be interesting to see which combinators were appropriate for an operational account of non-primitive actions. As an example compare the following two English language sentences:

- i Mary is permitted to write or post a letter.
- ii Mary is permitted to write a letter or Mary is permitted to post a letter.

In i the scope of the permission operator seems to be around a choice of actions, whereas in ii the choice is between two permissible actions. Analysis regarding the relationship between such sentences and, more heatedly, their representation in deontic logic has been the subject of much debate, see [Jen86] for some good discussion on this. Should we, for instance, relate sentences such as i and ii by axioms of the following form:

- iii $\text{Permission}(\text{Mary}, \text{'Write or Post'})$
 $\leftrightarrow \text{Permission}(\text{Mary}, \text{'Write'}) \text{ or } \text{Permission}(\text{Mary}, \text{'Post'}) ?$

The problem is compounded by the fact that the English language use the disjunctive *or* and the conjunctive *and* in many different ways (as we suspect do other natural languages). It would be interesting to see how such operational connections could be made in DL.

Last, but by no means least, our framework provides only an implicit treatment of time. Although one could argue that the temporal dimension of a system could be expressed by using the idea of clocks (time measurers, performing time marking actions such as the "tick-tock" of a clock) as agents, this, due largely to the fact that agents may not perform concurrently, is less than satisfactory. This approach becomes even less acceptable when we observe that actions themselves take time to occur. Indeed, it is this fact that actions are not instantaneous, which leads to the difficulties found in specifying concurrent systems. One only has to look at the large amount of work, see for instance [HO80] or [D-H86] that has been done in this area to see that this is a non trivial problem.

It seems to us that the approach which offers the most, both in terms of the "natural" way in which it models time and its expressiveness, is based around the concept of intervals. The idea is that an interval should represent a specific quantity of time, in which a property holds true. Actions are then attributed to properties such as "Happening" or "Occurring" and are said to "Happen" or "Occur" during an interval. Rather than explore this further, we refer the reader to either [KM88] or [Ma86] in which a temporal extension of our framework may be found.

This temporal interval based extension and allows our framework to cope with the specification of concurrent actions which overlap (in the temporal dimension) with each other in a variety of ways. However, we still reject the idea of a "logical" global reference of time and prefer instead the introduction of system defined "clocks" which generate (by performing actions) properties that signify the marking of appropriate quantities of time. These "clocks" (there may be more than one) can then be used to regulate other system behaviour.

One reason for not including this temporal extension here is that it is still under development. To be specific, it has only been defined at a semantical level and currently lacks a corresponding proof theory. The reader is referred to [KM88] or [KM88a] for our current views and progress on this. In the next section we shall look at a number of ways in which the framework we have proposed has been exploited.

6.3 Exploitation.

So far we have described, in some detail, the shortcomings of our framework and the potential solutions to these. We would like to end on a more positive note by identifying those areas where our work has already been used to some advantage.

Protocol specification.

One use of the deontic notions incorporated in DL is to be found in [D-H86] wherein the author describes a top down refinement methodology for the creation of protocol specifications (for communication networks). Rather than look at the main thrust of this work we shall restrict our attention to the use of the deontic notions of permission and obligation in this context.

Broadly speaking the author identifies two areas in which these notions are useful if not essential. The first concerns the specification of what are called interface constraints and the second involves the description of what is called "global behaviour". The former is a class of requirements which characterise the "correct" use of a communications protocol, whereas the latter is concerned with specifying the inherent behaviour of the system. These areas are by no means exclusive and some interplay is to be expected.

Actions in the interface constraints context are referred to as service primitives (primitive actions which the user calls upon to communicate with the communication medium and other users. We can see an example of an interface constraint if we consider a public telephone service. Here it is commonly known that no call can be made unless the telephone receiver has been lifted or money inserted. We saw examples of this in chapter five where we discussed a telephone network system. Thus, interface constraints state under which circumstances users are permitted (and indeed obliged) to invoke service primitives. This idea can also be seen in the more traditional context of data structures; a good example of this is the prohibition of popping an empty stack - see chapter two for more detailed arguments.

The deontic notions of obligation and permission are also used in specifying the functionality or behaviour of the system. The reader can see this from the network example of chapter four, and rather than reiterating what has already been covered there we shall look at the specific way in which obligations are used in [D-H86] to allow us to

ensure a certain robustness of some communication systems. The example given in [D-H86] concerns a token-passing protocol, so called because use of the network is restricted to only one "node" on the network (the one in possession of the token). When a node no longer has any use of the network the token is passed to some other node (usually in some predetermined order), hence all nodes will get an opportunity to use the network. But suppose for the sake of argument that some node "breaks down" whilst in possession of the token. In such a situation it is usual to wait for some fixed period of time after which the failed node is assumed inoperative and some other node elected and given the token. What we are leading up to here is that the newly elected node is also *obliged* to use the network (to show that it is operative) and so start the ball (token) rolling again. If this fails to happen (within the allowed time) the system could mistakenly assume that this node is also inoperative. Note that, here, the force of obligation is required to ensure that the node in question does at least attempt to pass on the token. Hence, if the node still fails to pass on the token it can be assumed with some certainty that the node is inoperative.

In conclusion we should point out that the underlying state transition mechanism and hence the characterisations of permissions and obligations in [D-H86] are not those which we have described in this work. We believe that this highlights the strength of intuition which underlies these deontic notions. To some extent, as we have already argued in chapter three, the precise nature of permission and obligation should and does change according to context. What matters most is their very broad application in almost all aspects and all levels of computer science.

Dialogue specification.

This area of specification is becoming increasingly significant in the design of computer embedded environments. The term dialogue is used to express communication between a number of entities: between man and machine, between machines (conceptual or physical) and indeed even between humans. The majority of work in this area has adopted the state transition view of communication, the analysis of speech acts being a particularly good case in point. Here dialogue is seen to be an interchange of information between two or more entities (usually humans), the basis for the communication being acts of speech. A good survey and introduction to speech act theory may be found in [SKB80].

The speech act approach to dialogue fits well with the style of specification approach we

are advocating in this work. This can best be seen by considering the two major components of a dialogue: a communication medium (by which to exchange information) and some form of control or usage rules by which to control the communication medium. For the purpose of dialogue specification we can identify a communication medium with the description of communication actions (those which enable an exchange of information between agents). Once the basis of communication has been defined in this way it can be harnessed by prescribing the use of communication actions (using the deontic component of DL) to specify meaningful and purposeful dialogue.

A good example of the use of this style of specification can be found in [IT86] where the authors were commissioned by ITT (International Telephone & Telegraph) to first capture the requirements and subsequently design a CBT (Computer Based Training) package for packet switched network operators. The general approach taken in this study is based on the traditional CBT technique of first displaying and then testing the student's understanding of certain application knowledge. The end product needed to be flexible enough to deal with students of varying ability and hence required an evaluation and score keeping mechanism.

The authors of [IT86] used a precursor of DL in order to specify the CBT package. The interaction between potential student and tutor (CBT system) was viewed as a controlled dialogue. The first order predicate component of DL was used to specify the basic concepts of the application domain (packet switched networks). Action descriptions were used to describe the basic means of tutor/student interaction. Examples of this include specific display and test actions performed by the tutor (modelled as an agent) and erotetic and response actions by the student (also modelled as an agent). Finally, the deontic component of DL was used to control the student/tutor dialogue, specifying when a student is permitted/obliged to respond to the tutor, which in turn may permit or oblige the tutor to perform some further action. This study [IT86] was particularly rewarding as ITT went on to prototype the training system using micro-Prolog [CM84] straight from the DL specification.

Another use of DL in a dialectical context can be found in [Jer87]. Here, the author is concerned with modelling dialogue using game theoretic notions. Again the approach taken is to use the prescriptive component of DL to model the game theoretic ideas of opponent and proponent interaction. One aim of this is to use the resulting specifications and the logical apparatus of DL to prove that certain dialectic games possess associated

consequence relations as part of their proof theory.

This strategy of modelling dialogue has already had some success by modelling the traditional game theoretic interpretation of the classical and intuitionistic first order systems. The interested reader is referred to [Jer87] for further details.

Finally, and perhaps tangentially, we should also like to mention work on natural language [Pit88]. Work is currently in progress to use DL within the (intensional logic) tradition of Montague's semantics [DWP81] of natural language. DL will be used to represent a (necessarily) restricted but expressive subset of the English language. Briefly, the use of DL as a representation of the meaning of English expressions will induce either a model or proof theoretic interpretation of such expressions. One of the benefits of this approach is the ability to analyse English expressions through the formal machinery of DL.

6.4 Research Collaboration.

One project which has adopted many of the ideas and much of the spirit of DL is the FOREST project. FOREST, which stands for "Formal Requirement Specification Techniques", is an Alvey sponsored collaboration between Imperial College, GEC Avionics, GEC Marconi Research Centre and the Harwell Laboratory.

The main aims of the project are threefold: develop a formal language for the specification of real time, embedded computer systems; create an associated method to elicit and validate such requirements; and build tools to aid the process of both eliciting and formulating such requirements.

The formal language developed by FOREST is a modal logic, very similar to DL, called MAL (standing for Modal Action Logic). As recognised in [Kho87] much work needs to be done if this is to become a *user friendly* specification language. Both syntactic sugaring and more fundamental structuring constructs need to be introduced if the language is to achieve success in the industrial workplace. The process of structuring is currently being tackled by considering the structure of the associated method for eliciting requirements.

The FOREST methodology for eliciting requirements is known as SCS (Structured Common Sense) and is targeted specifically on the MAL formal system. The method itself uses a number of well known industry standard techniques, such as data flow diagrams and E-R-A (Entity-Relationship-Attribute) diagrams, in order to progressively build up a picture of a system's requirements. Much of these techniques are introduced to form an infrastructure for guiding and organising the elicitation activity by which a formal specification is obtained (from an invariably informal application concept). The details and background (including a survey of extant requirement elicitation methodologies) leading up to the development of SCS can be found in [FP87].

Tool support for both SCS and MAL is still in its infancy and is a priority for further development. As a first attempt at providing tool support for MAL, a simulator was developed to animate some of the dynamic aspects of the language, the details of which can be found in [Boo86]. More recently, firmer foundations have been laid for tool development. The work in [CC87] describes a tableaux proof system for MAL, showing that MAL is both sound and complete with respect to a more conventional truth

functional possible worlds semantics. Work is currently in progress to develop a theorem prover for MAL based on this tableaux system.

Although the FOREST project has achieved a great deal, not all the original goals of the project have been reached. This is due, in part, to the ambitious nature of these goals in relation to the resources allocated to the project. However, it is encouraging that this work is planned to continue in several future research projects.

Another project worth mentioning here is the EEC COST 13 initiative, sub-project ii, entitled "Non Monotonic Logics for Knowledgebase Specification". The main purpose of this project was to facilitate a cross fertilisation between three research sites: Imperial College (London, England), CERT (Toulouse, France) and INESC (Lisbon, Portugal).

The main outcome of the first term of the project was a plan of action to bring together the complementary strands of work at each of the participating sites. All the work focused on the specification and design of knowledgebases. The work in Toulouse, described in [Cho86], characterised a strategy to control the frame problem. At the other end of the spectrum the deontic component of DL, Imperial College's contribution, was adopted as a means to specify the desired evolution of a knowledgebase. This was made possible by the project adopting the usual pre/post condition style of action description and using the prescriptive elements of DL to govern behaviour. Finally the work at INESC described in [Car85] and [FS86] suggested a multi layered approach to the design of temporal and other dynamic constraints.

The work is summarised in an end of project report [CCFK] submitted to the EEC. Further funding was requested and subsequently obtained and all three sites are actively pursuing the goals set by the first phase of the project.

Finally, having taken so much from the philosophical arena, we feel obliged (no pun intended) to "give something back". We think that some of the ideas gained from using DL as a means of specifying computer systems can be used to refine and augment the formalisation of various deontic logics. Perhaps even extending to small contributions in ethical, moral and legal philosophy (the original fields of application). The last of which is already becoming an area of interest to computer scientists, see for instance [Sta80], [Ser85] and [SFS88], who are currently battling to represent and "automate" certain aspects of the law. We look upon this as a new dimension within which to extend and develop our work.

BIBLIOGRAPHY

- [AH85] G. Agha and C. Hewitt, Concurrent Programming using Actors: Exploiting Large Scale Parallelism, in in Proc. of 5th Conf. of Foundations of Software Technology and Theoretical Computer Science, pp 19-41, Pub. Springer-Verlag, ed. S. Maheeshwari, 1985.
- [ADJ77] J. A. Gougen, J. W. Thatcher, E. G. Wagner, J. B. Wright, Initial Algebraic Semantics and Continous Algebras, JACM vol 24, No. 1, pp 68-95, 1977.
- [Ba81] J. Barron, Dialogue and Process Design for Interactive Information Systems Using TAXIS, Tech. Rep. CSRG-128, Univ. of Toronto, 1981.
- [Boo86] J. Booth, A Simulator for MAL (Modal Action Logic), Alvey FOREST Deliverable Report , GEC Research Laboratories, Marconi Research Centre, Great Baddow, Chelmsford, England, 1986.
- [Bor81] A Borr, Transaction Moinitoring in Encompass-Reliable Distributed Transaction Processing, in Proc. 7th VLDB IEEE, New York, 1981.
- [BG80] R. M. Burstall, J. A. Goguen. "The Semantics of CLEAR, a Specification Language", Proc. of *Advanced Course on Abstract Software Specifications*, Copenhagen, LNCS86, Springer-Verlag, 1980.
- [BMS80] R. Burstall, D. MacQueen and D. Sannella, HOPE: An Experimental Application Language, Internal Report CSR-62-80, Edinburgh University, Dept. of Computing, 1980.
- [Car85] J. Carmo, "The INFOLOG Branching Logic and Events", IFIP WG8.1, Proc. Working Conference on Theoretical and Formal Aspects of Information Systems, Sitges, 1985.
- [Cas72] H-N Castaneda, "On the Semantics of Ought-to-do", in *Semantics of Natural Language* , pp 675-694, eds Davidson & Harman, D. Reidel Publishing Company, 1972.

- [Cas82] H-N Castaneda, The Logical Structure Of Legal Systems: A New Perspective, in Deontic Logic and Legal Reasoning, vol II, ed. A.A. Martino, Pub North-Holland, 1982.
- [Che80] B. F. Chellas, Modal Logic an Introduction, Cambridge University Press, Cambridge, 1980.
- [Cho86] L. Cholvy, Update Semantics Under The Domain Closure Assumption, in Proc. of ICPT'86, Rome, 1986.
- [CC87] J. Cunningham & M. da Costa, Computational Logic and Modal Action Logic, Alvey FOREST Deliverable Report , GEC Research Laboratories, Marconi Research Centre, Great Baddow, Chelmsford, England, 1987.
- [CCF81] M. A. Cassanova, J. M. V. Castilho, A. L. Furtado, Properties of Conceptual and External Database Schema, Series: Monografias em Ciencias da Computacao, No 11/81, Departamento de Informatica P.U.C. do Rio de Janerio, 1981.
- [CCFK] J. Carmo, L. Cholvy, J. Fiadeiro and S. Khosla, End of Project Report, COST 13, "Non Standard Logics For Automatic Reasoning", sub-project ii, EEC.
- [CL85] P. Cohen and H. Levesque, Speech acts and rationality, in Proc. of the 23rd Conf. of the Assoc. for Computational Linguistics, pp 49-59, Stanford Ca., 1985.
- [CM84] K. Clark and F McCabe, micro-PROLOG: Programming in Logic, Pub. Prentice-Hall, International Series in Computer Science, 1984.
- [Dij76] E. W. Dijkstra, *A Discipline of Programming* , Pub. Prentice-Hall, 1976.
- [D-H86] L. Da-Hai, Top-down and Step-wise Refinement of Protocol Specifications, Phd Thesis, Imperial College, London University, 1986.

- [DMW80] W. Dosch, G. Mascari, M. Wirsing, "On the Algebraic specification of databases", Proc. 8th International Conference on Very Large Data Bases (VLDB), pp 370-385, 1982.
- [DWP81] D. Dowty, R. Wall and S. Peters, "Introduction to Montague Semantics", in Synthese Language Library, D. Reidel Publishing Company, 1981.
- [Eh82] H-D. Ehrich, On the Theory of Specification, Implementation and Parameterisation of Abstract Data Types, JACM, vol 29, No. 1, 1982.
- [En72] H. B. Enderton, "A Mathematical Introduction to Logic", Academic Press (Pub.), 1972.
- [Flo67] R. W. Floyd, Assigning Meanings to Programs, *Proceedings of the Symposium of Applied Mathematics*, vol 19, J. T. Schwartz (ed), American Math. Soc., Providence, 19 - 32, 1967.
- [FH81] D. Føllesdal and R. Hilpinen, "Deontic Logic: An Introduction", pp 1-35, in Deontic Logic: Introductory and Systematic Readings, ed R. Hilpinen, D. Reidel Publishing company, 1981.
- [FH85] R. Fagin and J. Y. Halpern, Belief, Awareness and Limited Reasoning, in Proc. of 9th IJCAI, pp 491-501, 1985.
- [FP87] A. Finkelstein & C. Potts, "Structured Common Sense: An Elicitation Methodology for the FOREST Formal System (MAL)", FOREST (ALVEY Project) Report, 1987.
- [FN71] R.E. Fikes and N.J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", in Artificial Intelligence, vol 2 pp 189-208, 1971.
- [FS86] J. Fiadeiro and A. Sernadas, "The INFOLOG Linear Tense Propositional Logic of Events and Transactions, in Information Systems 11(1), 1986.
- [Go82] J. A. Goguen, Merged Views, Closed Worlds and Ordered Sorts, Internal Report, SRI, 1982.

- [Go87] S. Goldsack, A Tutorial on the FOREST Formalism, FOREST (ALVEY project) Report, 1987.
- [Gol82] R. Goldblatt, *Axiomatizing the logic of Computer Programming*, LNCS 130, Springer-Verlag (Pub.), 1982.
- [Gu77] J. V. Guttag, Abstract Data Types and the Development of Data Structures, *Communication of ACM*, vol 20, No. 6, pp. 396-404, 1977.
- [GH78] J. V. Guttag and J. J. Horning, The Algebraic Specification of Abstract Data Types, *Acta Informatica*, vol 10, No. 1, pp. 27-52, 1978.
- [GMS83] F. Golshani, T.S.E Maibaum, M.R. Sadler, A modal System of Algebras for Database Specification and Query/Update support, Proc.9th International Conference on Very Large Databases, Italy, 1983.
- [He69] C. Hewitt, "Planner: A language for Proving Theorems in Robots", in Proc. IJCAI, Washington DC, pp 295-301, 1969.
- [He84] C. Hewitt et al, Linguistic Support of Receptionists for Shared Resources, in Proc. NSF/SERC Seminar on Concurrency, Pub. Springer-Verlag, 1984.
- [Hen82] M. Hennessy, Synchronous and Asynchronous Experiments on Processes, Internal Report CSR-125-82, Univ. of Edinburgh (U.K.) 1982.
- [Ho69] C. A. R. Hoare, An Axiomatic Basis for Computer programming, *Communication of ACM* 21, Oct. 1969.
- [Ho72A] C. A. R. Hoare, Proof of Correctness of Data Representations, *Acta Informatica*, vol 4, pp. 271-281, 1972.
- [Ho72B] C. A. R. Hoare, Notes on Data Structuring, in Structured Programming, by O-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Academic Press.

- [Ho83] C. A. R. Hoare, Notes on Communicating Sequential Processes, Technical Monograph PRG-33, Programming Research Group, University of Oxford, 1983.
- [HC68] G. Hughes and M. Cresswell, An Introduction to Modal Logic, Pub. by Methuen, London, England, 1968.
- [HGQ87] P. Howells, S. Goldsack, B. Quirk, "Designing From Modal Action Logic", FOREST (ALVEY project) Report, 1987.
- [HO80] B. T. Hailpern and S. Owicki, Verifying Network Protocols Using Temporal Logic, Technical Report 192, Dept. of Computer Science, Stanford, June, 1980.
- [Ich79] J. D. Ichbiah et al, Rationale for the design of the ADA programming language, SIGPLAN Notices 14,6, part B, 1979.
- [IT86] S. Khosla and C. Potts, "Deontic Action Logic as a Formal Basis for an Intelligent Tutoring System", ITT Report, 1986. Also available from the authors at Imperial College London.
- [Jen86] R. E. Jennings, "A Natural Deontic Logic", Journal of the Philosophy of Logic, 1986.
- [Jer87] P. Jeramaes, Dialectical Games, Internal Report, Dept. of Computing, Imperial College, London, 1987.
- [Jo80] C. B. Jones, Software Development: a Rigorous Approach, Prentice-Hall International Series in Computer Science, 1980.
- [Jo86] C. B. Jones, Systematic Software Development using VDM, Prentice-Hall International Series in Computer Science, 1986.
- [Jon87] A. J. I. Jones, Deontic Logic And Its Implementation: Some Preliminary Comments, Private Notes, Dept of Philosophy, University of Oslo.

- [JKM86] P. Jeremaes, S. Khosla and T. Maibaum, "A Modal (action) Logic for Requirements Specification", in *Software Engineering 86*, eds. P. Brown and D. Barnes, Pub. Peter Peregrinus, 1986.
- [Kho86] S. Khosla, FOREST: Technology Transfer, Alvey FOREST Deliverable Report, GEC Research Laboratories, Marconi Research Centre, Great Baddow, Chelmsford, England, 1986.
- [Ko79] R. Kowalski, *Logic for Problem Solving*, North-Holland, N. Y., 1979.
- [KM88] S. Khosla and T. Maibaum, "Time, Function and Behaviour", in Workshop on Temporal Logic (to appear), ed H. Barringer, Pub. Sprinver-Verlag, 1988.
- [KM88a] S. Khosla and T. Maibaum, "Formal requirements of Real Time Systems", in Specification Techniques for Concurrent Systems (to appear), eds H. Barringer, A. Hill, B. Kreig-Brucker, Pub. Sprinver-Verlag, 1988.
- [KMS85] S. Khosla, T. Maibaum and M. Sadler, "Large Database Specifications from Small Views", in Proc. of 5th Conf. of Foundations of Software Technology and Theoretical Computer Science, pp 246-271, Pub. Springer-Verlag, ed. S. Maheeshwari, 1985.
- [KMS86] S. Khosla, T. Maibaum and M. Sadler, "Database Specification" in Database Semantics (DS-1), proc. IFIP conf. on Database Semantics, Ed. by Steel and Meersman, Pub. by North-Holland 1986.
- [Lis77] B. Liskov et al, Abstraction mechanisms in CLU, in *Comm. ACM* 20/8, pp 564-576, 1977.
- [Le86] R. Lee, A Denotational Semantics for Administrative Databases, in Database Semantics (DS-1), proc. IFIP conf. on Database Semantics, Ed. by Steel and Meersman, Pub. by North-Holland 1986.
- [Lem65] E. J. Lemmon, *Begining Logic*, pubs Nelson, London and Edinburgh, 1965.

- [LEG85] U. Lipeck, H-D. Ehrich and M. Gogolla, "Specifiying Admissibility of Dynamic Database Behaviour Using Temporal Logic", in Theoretical and Formal Aspects of Information Systems, pp 145-157, eds Bubenko et al, Pub. North-Holland, 1985.
- [LL81] M. Leonard and B. T. Luong, "Information Systems Design Approach Integrating Data and Transactions", Proc. 7th VLDB IEEE, pp 235-246, New York, 1981.
- [LN] M. Lewis and D. Nute, "The Automation of a Practical Reasoning System Based on Concepts in Deontic Logic", Internal Report, Advanced Computational Methods Center, Univ. of Georgia, Athens, Georgia, USA.
- [LZ75] B. Liskov, S. N. Zilles, "Specification Techniques for Data Abstractions", IEEE Transactions on Software Engineering, vol 1, No. 1, pp 7-19, 1975.
- [Ma84] T. Maibaum, "Database Instances, Abstract Data Types and Database Specification, in the Computing Journal of the British Computer Society, vol 28, No. 1, 1984.
- [Ma86] T. Maibaum, "A Logic for the Formal Requirements Specification of Real-Time/Embedded Systems", Alvey FOREST Deliverable Report 3, GEC Research Laboratories, Marconi Research Centre, Great Baddow, Chelmsford, England, 1986.
- [Mac] Macintosh is a registered trademark of the Apple (🍏) Computer Corporation USA.
- [McC83] L. T. McCarty, Permissions and Obligations, in Proc. IJCAI-83, pp 287-294, 1983.
- [Mi80] R. Milner, A Calculus of Communicating Systems, in LNCS 92, Springer-Verlag Pub., 1980.
- [Mo84] C. C. Morgan, Schemas in Z - A Preliminary Reference Manual, Programming Research Group, University of Oxford, 1984.

- [M-L79] Per Martin-Lof, Constructive Mathematics and Computer Programs, Proc. 6th Int. Congress for Logic Methodology and Philosophy of Science, Hanover, 1979.
- [ML85] N. Minsky and A. Lockman, Ensuring Integrity by Adding Obligations to Priveleges, in Proc 8th IEEE Intl. Conf. on Software Engineering, 1985.
- [MSF80] T. S. E. Maibaum, C. S. dod Santos, A. L. Furtado, A Uniform Logical Treatment of Queries and Updates, Technical report DBO18001, Departamento de Informatica P.U.C. do Rio de Janerio, 1980.
- [MSV85] T. Maibaum, P. Veloso and M. Sadler, "A Theory Of Abstract Data Types For Program Development: Bridging The Gap", in Formal Methods and Software Development, LNCS 186 vol 2, TAPSOFT, eds G. Goos and J. Hartmanis, Pub. Springer-Verlag, 1985.
- [MW85] Z. Manna and R. Waldinger, The logical Basis for Computer Programming Volume I: Deductive Reasoning, Pub. by Addison-Wesley, 1985.
- [Nute] D. Nute, "PROWIS, An Executable Language for Defeasible Reasoning", Internal Report, Advanced Computational Methods Center, Univ. of Georgia, Athens, Georgia, USA.
- [NHN80] R. Nakajima, M. Honda and H. Nakahura, Hierarchical Program Specification and Verification - A Many-sorted Approach, in Acta Informatica, vol 14, 1980.
- [N-S85] W.H. Newton-Smith, Logic: An Introductory Course, Pub. Routelege & Kegan Paul, 1985.
- [Pe79] C. A. Petri, "Concurrency", in LNCS 84, Springer-Verlag Pub., 1979.
- [Ped86] E. Pednault, "Formulating Multi-Agent, Dynamic-World Problems In The Classical Planning Framework", in in Reasoning about Actions & Plans, Proc. 1986 workshop, Timberline Oregon, pp 47-82, 1986.

- [Pit88] J. Pitt, The Use of MAL (Modal Action Logic) as a Representation Language for Natural Language Expressions, (ALVEY) FOREST Report, Dept. of Computing, Imperial College, London, 1988.
- [Pnu77] A. Pnueli, "The temporal Logic of Programs", Proc. 18th Annual ACM Symposium on Foundations of Computer Science, IEEE, pp 45-57, 1977.
- [Pol86] M. Pollack, A Model of Plan Inference that distinguishes between the Beliefs of Actors and Observers, in Reasoning about Actions & Plans, Proc. 1986 workshop, Timberline Oregon, pp 279-296, 1986.
- [Pra83] V. R. Pratt, Five Paradigm Shifts in Programming Language Design and their Realisation in Viron, a Dataflow Programming Environment, in Proc. 10th ACM Conf. on Principles of Programming Languages, 1983.
- [Rei78] R. Reiter, "On Closed World Data Bases", in Logic and Data Bases, pp 55-76, eds Gallaiere and Minker, Pub. Plenum Press, New York, 1978.
- [Rei80] R. Reiter, "A Logic for Default Reasoning", *Artificial Intelligence*, 13, pp 81-132, 1980.
- [RU71] N. Rescher and A. Urquhart, Temporal Logic, Pub. Springer-Verlag, 1971.
- [Sad88] M. Sadler et al, A framework For Dialectic Studies, Internal Report, Imperial College, Dept. of Computing, London, 1988.
- [Ser85] M. Sergot, "Representing Legislation as Logic Programs", in Machine Intelligence 11, 1985.
- [Sh86] Y. Shoam, *What is the Frame Problem ?*, in Reasoning about Actions & Plans, Proc. 1986 workshop, Timberline Oregon, pp 83-98, 1986.
- [Small] Smalltalk80, Reference Manual, Xerox Parc, Palo Alto Labs, USA.
- [Sta80] R. Stamper, "Legol: Modelling Legal Rules by Computer, in Computer and Law, ed. B. Niblett, Cambridge University Press, 1980.

- [Su84] B.A. Sufrin, "Notes for a Z Handbook: Part I - The Mathematical Language", Programming Research Group, University of Oxford, 1984.
- [SFS88] C. Sernades, J. Fieaderio and A. Sernades, "Object Oriented Coneptual Modelling, in The Role of A.I. in Databases and Information Systems, eds Kung and Meersman, Pub North-Holland, 1988.
- [SK86] A. Solvberg and C. Kung, "On Structural And Behavioural Modelling of Reality, in Database Semantics (DS-1), proc. IFIP conf. on Database Semantics, Ed. by Steel and Meersman, Pub. by North-Holland 1986.
- [SKB80] J. Searle, F. Kiefer, and M. Bierwish (eds.), Speech Act Theory and Pragmatics, in Synthese Language Library, D. Reidel Publishing Company, 1980.
- [SW82] D. Sanella, M. Wirsing, Implementation of Parameterised Specifications, 9th ICALP, LNCS140, Springer-Verlag, 1982.
- [Tal85] J. Talja, On The Logic Of Omissions, in *Synthese* 65, pp 235-248, by D. Reidel Publishing Company, 1985.
- [Vi86] Steve Vickers, *Private Communication*, Imperial College, 1986.
- [VCF84] P. Veloso, M. Cassanova and A. Furtado, "Formal Data Base Specification An Eclectic Perspective" Series: Monografias em Ciencias da Computacao, No 1/84, Departamento de Informatica P.U.C. do Rio de Janerio, 1984.
- [We79] H. Weber, Modularity in Data Base System Design, in Current Trends in Data Base Systems, Eds. H. Weber & A. Wassermann, Pub. by North Holland, 1979.
- [We83] H. Weber, Object Oriented DDBS Design, in Proc. of the second Intl. Conf. on Databases, (ICOD-2), Eds. S. Deen and P. Hammersley, Pub. by Wiley for The British Computer Society, 1983.

- [Win83] J. Wing, A Two Tiered Approach to Specifying Programs, Ph.D. Thesis, Laboratory for Computer Science, MIT/LCS/TR-299, 1983.
- [Wir71] N. Wirth, The Design of A Pascal Compiler, in *Software - Practice and Experience I*, 4 (Sep), pp 309-333, 1971.
- [Wir83a] M. Wirsing, Structured Algebraic Specifications: A Kernal Language, PhD Thesis, Munchen, Germany, 1983.
- [Wir83] M. Wirsing et.al., On Hierarchies of Abstract Data Types, *Acta Informatica*, vol 20, No. 1, pp 1-33, 1983.
- [Wri51] G.H. von Wright, *An Essay in Modal Logic* , North-Holland Publishing Company, Amsterdam, 1951.
- [Wri63] G.H. von Wright, Norm and Action, Routledge and Kegan Paul, London, 1963.
- [Wri80] G.H. von Wright, "Problems and Prospects of Deontic Logic: A Survey", in *Modern Logic - A Survey: Historical, Philosophical and Mathematical Aspects of Modern Logic and Its Applications*, pp 399-423, D. Reidel Publishing Company, ed. E. Agazzi, 1980.
- [Wri81] G.H. von Wright, " On the Logic of Norms and Actions", in *New Studies in Deontic Logic*, D. Reidel Publishing Company, ed. R. Hilpinen, 1981.
- [Wri82] G.H. von Wright, "Norms, Truth and Logic", in *Deontic Logic and Legal Reasoning*, vol II, ed. A.A. Martino, Pub North-Holland, 1982.
- [Zi73] S. N. Zilles, "Algebraic Specifications of Data types", in Project MAC progress report for 1973-74, CSG Memo 119, MIT, Cambridge, Mass.pp 1-12, 1973.