DEPARTMENT OF COMPUTING
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE
UNIVERSITY OF LONDON

# Proof-Theoretic Foundations for the Design of Extensible Software Systems

*Carlos Henrique Cabral Duarte*

A thesis submitted to the University of London
in partial fulfilment of the requirements for
the degree of Doctor of Philosophy (PhD) and
the Diploma of the Imperial College (DIC)

(1998)

# Abstract

Extensible software systems have been increasingly demanded as a means of supporting in a more faithful way constantly changing user requirements and also as a necessary logical counterpart to rapidly evolving networking architectures. Such terms as open, reconfigurable, mobile and reflexive have been used to attempt to describe relevant facets of this kind of reactive system with dynamically varying functionality or structure. In this thesis, we not only characterise extensible systems but also study their rigorous design.

We advocate a proof-theoretic step-by-step approach to the development of extensible systems as a means of ensuring correctness, modularity and incrementability. By spelling out their characteristics and identifying corresponding logical constructions, we present as an original foundational contribution a first-order branching time logical system that seems to be appropriate as a basis for specification and verification. Even though our software process approach is proof-theoretic, we provide both model and proof theories for the proposed system, studying in the context of general logics important properties such as soundness, completeness and expressiveness. We argue that other logical systems proposed in the literature are not adequate to achieve the same desirable effects in design.

We also study particular software development approaches based on the actor model, on dynamic sub-classing and on meta-level architectures which could best underpin the rigorous design of extensible systems. Specific design principles are proposed in the form of derived inference rules with their application guidelines and composability notions are studied in terms of categories of theory presentations. We show that reasoning about their local properties can be carried out based only on such constructions but global properties may not be verified without the additional aid of a rely-guarantee discipline. A series of helpful theorems and realistic examples are developed to support and illustrate how our ideas can be effectively applied in practice.

*To my family*

# Acknowledgements

I would like to thank Professor Tom Maibaum, the supervisor of the research leading to this thesis, for his guidance, incentive and for providing physical and additional financial resources to support my research whenever necessary.

In addition, I wish to acknowledge the support offered by the following people, ranging from single words of encouragement to technical comments on my work, all of which of fundamental importance to push forward my research: Gul Agha, María Victoria Cengarle, Adrian Chung, Stefan Conrad, Theodosis Dimitrakos, José Fiadeiro, Marcelo Finger, Fausto Giunchiglia, Vassos Hadzilackos, Ian Hodkinson, Kamyar Kanami, Marta Kwiatkowska, Ezra Mugisa, Raja Nagarajan, Valéria de Paiva, Dimitris Raptis, Mark Ryan, Prahladavaradan Sampath and Carolyn Talcott.

Androulla Pieri, E. Ann Halford, Denise Grant and Barbara Claxton provided excellent administrative services during my stay at the Department of Computing. John Hughes and Hillary Glasman-Deal were of invaluable assistance in improving my spoken and written English language skills through the classes of the Humanities Programme. I would like to thank them all for their help and for always maintaining a friendly working environment.

The development of this work at the Imperial College was financially supported in full by CNPq, the Brazilian National Research Council, research grant number 200766/94-5. I am also grateful to the Brazilian National Bank of Social and Economic Development (BNDES) for allowing my stay in the United Kingdom for almost 4 years on leave of absence.

During these years, it would have been impossible to advance in my personal project without the support and comprehension of Patrícia, Matheus and Carolina. I cannot quantify the numerous occasions in which I unfortunately had either to leave your enjoyable leisure company or to ignore completely many of our problems in order to concentrate on my studies alone. This thesis is dedicated to you, my beloved family. All these years could also have been more pleasant for Aracy, Gisele and Thais, my dear mother, sister and niece, if the physical distance separating us were not as huge. I especially wish to thank you for your understanding of how important it was for me to conclude this work.

# Table of Contents

# List of Figures

# Chapter 1

# Introduction

This thesis is about theoretical foundations for the design of extensible software systems. This means that we are interested in providing here a characterisation of extensible systems as well as studying formal theories to support their rigorous specification and verification. As such, the thesis can be regarded as the outcome of research in three distinct subject areas: Theory of Computing, Software Engineering and Distributed Systems.

The pressing need to support distributed extensible systems has recently appeared as a result of technological innovation. At the current moment, it is possible to use portable computers, cellular phones, personal digital assistants and other devices connected to worldwide networks, which are in this way sparsely distributed and fairly heterogeneous. Because the interconnections between these hardware components may change at any time and it is normally possible to attach new equipment to the network and disconnect some of its parts, it seems to be reasonable to consider this architectural style as extensible. The required software components that populate these machines may in turn be remotely used, created and reconfigured. Moreover, it is often the case that such components can move from one node of the architecture to another. End users correctly perceive these movements through revisions in the functionality provided at their current location. Software systems organised in this way as well as some of their sub-systems can again be regarded as extensible.

The same sort of software system is desirable for other reasons if we examine their engineering process. Clearly, to design, implement, test and make a software system available for use may take sufficient time to allow the initial requirements to change in perhaps unpredictable ways. In those cases where it is feasible to design the system so that it can be dynamically altered according to some particular customer needs, such solution appears to be more convenient because it may avoid maintenance. Depending on the way the system was designed, modifications may be produced by agents such as the end user, the (meta-level) objects present in the operational environment and so on, and many may be the methods supporting this process of change, by interacting with an appropriate sub-system or by using the whole system to refine a model causally connected to its own behaviour, for example. These methods may alter the current functionality and structure of the system to such an extent that yet again software systems with these characteristics can be regarded as extensible.

It is not difficult to figure out that the characteristics above turn the development of extensible systems into an activity even more difficult and error prone than usual. It is well-known on the one hand that, given a set of requirements, the unique way of ensuring the correctness of an implementation with respect to these requirements, meaning that no errors were introduced throughout the development of the system, is to adopt a (set of) logical system(s) and use formal, theoretical constructions to prove that the implementation satisfies the specification of the requirements and is therefore a valid realisation of the system. To verify that intuitive properties follow from a set of specifications also increases confidence in the adequacy of each proposed design. On the other hand, the step-by-step, systematic development of extensible systems presents its own peculiarities, which appear to demand particular logical systems to allow if not a formal at least a rigorous treatment. We devote this thesis to the study of isolated designs in this process and their formal theories.

## 1.1 What is Extensibility?

It should be evident at this point that extensibility is intrinsically related to the possibility of change. It has been claimed since the early days of Software Engineering that the right way of dealing with change throughout the development process is to anticipate them as much as possible (Parnas 1978). In effect, extensibility is an outcome of anticipation. To classify the distinct types of change software artifacts and related objects may suffer appears to be necessary here.

### 1.1.1 A Classification of Software Changes

The occurrence of changes throughout the life cycle of a system can affect two distinct kinds of entity: specifications during design and the state of both system and environment after deployment. *Static* changes, which affect a system description, are classified into *endogenous* and *exogenous* by Lehman *et al.* (1984) depending on the origin of the request for changes. If a change is required due to decisions made during the design, perhaps because they have made the continuation of the process impossible, the change is regarded as endogenous. Otherwise, if the change is caused by a modification in customer requirements, it is considered to be exogenous. While static changes oblige the designer to backtrack in the project, *dynamic* changes are a result of system behaviour.

According to this classification, it is possible that some change be regarded as both static and dynamic. For example, if a system keeps a model causally connected to (part of) its own behaviour and allows this model to be changed at run time, the description of the system will have changed as well as its behaviour after some modifications in the model. An example of this functionality is presented by the text editor Emacs (Stallman 1981). Software systems written in interpreted languages and reflective software architectures provide other real examples of this kind. We shall return to these examples in the sequel.

There is an additional classification of dynamic changes which is often useful in describing the properties of software systems. A change is said to be *functional* whenever it results in some modification in the functionality provided by the system. In addition, the change is *structural* if it implies a reorganisation of the interconnections between components of the system. Depending on the objects affected by a change, it is again possible to classify the same change in both categories. For example, in a telecommunications network, if a calling-number paging service becomes available whenever a call-forward service is not accessible, as a result of applying this rule part of the network must face a structural change whereas the whole system will have suffered a functional change. For a software system to be really extensible, to support some of these two types of change is a necessary requirement. As a corollary of this imperative, we obtain that purely functional programs cannot be extensible as it is impossible to capture notions of state and change in this way.

A characterisation of extensibility can be derived from the allowed degree of dynamic changes. We say that a software system is *customisable* whenever dynamic changes range only over second class entities such as constants from a fixed set. Conversely, a system is said to be *extensible* if changes also encompass first class objects, which are dynamically created, altered and referenced. For instance, if a Lisp program may only read configuration files not containing function definitions, if some concurrent processes only admit a fixed set of configurations, these systems are considered to be customisable. Otherwise, they are regarded as extensible. Bearing in mind this definition, it is easy to understand why Agha (1986) regards *openness* as a prerequisite for extensibility: without considering the existence of an environment and the ability to interact with other similar components therein, a system cannot be regarded as extensible.

### 1.1.2 Related Terminology

In order to further clarify the notion of an extensible software system, let us examine other terms which may at first seem to be directly related to extensibility but in fact refer, as they are defined in the literature, to many distinct stages of the development process.

The terms *adaptable* (Alencar *et al.* 1995) and *adaptive* (Lieberherr *et al.* 1994) have been both used to stand in more or less detail for a software development technique whereby *software artifacts*, specifications and implementations, are defined in a generic way so as to allow further particularisation, which may turn out to be subsequently necessary. Depending on whether or not there exists a systematic method for deriving particular instances from each generic description, the term adaptive is used. In both cases, the main focus of attention is in obtaining artifacts to serve as a practical basis for reuse in more advanced stages of the development of the same system or throughout the life cycle of other systems.

Lehman and Belady (1985) use *evolvability* to make reference to the property enjoyed by some systems of easily allowing maintenance. Kamel (1987) argues that this property is fundamentally related to the modular character of system components. Clearly, evolvability presupposes that some design steps have already happened and asserts how easy it is to backtrack in the process. Parnas (1978) in his paper was really referring to evolvable systems, proposing in addition techniques to ensure modularity and extensibility.

As an aside, it is important to mention that for historical reasons we have chosen to use here *extensible* as the flagship word to stand for the family of software systems we are interested in treating. The same term has been used by Matsuoka (1993) only to make reference to concurrent reflective object-based architectures and their features. It would certainly be incorrect in the context of this thesis to infer that, because we claim to be interested in dealing with extensible systems, to observe them presenting at some moment less functionality than in a previous instant would be forbidden. Of course, we strive to support equally not only the design of extension and contraction, being two faces of the same coin, but of any kind of dynamic change as well.

### 1.1.3 Approaches to Support Extensibility

Many ways of dealing with the design and implementation of software systems have been studied in the literature serving as means to guarantee extensibility. As a general rule, these approaches do not depend on any particular level of abstraction to be adopted and fall into one of the following categories:

**open reconfigurability:** Distributed systems consist in collections of loosely interconnected components. When such interconnections may vary at run time due to the addition of new components and as a result of changes in established connections, we say that the system is *reconfigurable*. Moreover, if it is possible for the system to interact at some point with an environment over which little if any control is kept, we say that the system is also *open*. The actor model initially proposed by Hewitt and Baker (1977) and later refined by Clinger (1981), Agha (1986) and Talcott (1997) appears to be the most faithful representative of this approach;

**dynamic sub-classing:** The notion of class is widely know within the object-based design community as defining collections of objects with the same behavioural characteristics (Wegner 1987). If it is possible for an object to migrate from one class to another at run time, we say that *dynamic sub-classing* is supported. This should not to be confused with *inheritance*, which is a reuse technique based on the hierarchical organisation of object descriptions. A detailed formal treatment of dynamic sub-classing has been developed by Wieringa *et al.* (1995);

**meta-architectures:** Computational objects are defined in terms of a set of primitive notions. Provided that it is possible for some objects to manipulate (a number of) these notions as if they were conventional data objects, we say that *meta-level* facilities are supported by the architecture. The most general case of meta-level support is that of *computational reflection*, wherein each object carries a description of its own behaviour and behaves in a way causally connected to such a description (Maes 1987).

The approaches above are based on distinct notions and give rise to extensible systems with diverse features. Not all of them are fully compatible with the conventional concept of rigorous stepwise development. In the following chapters, we shall study how to design systems in some of these ways, clarifying the reasons for regarding the others as unsuitable.

## 1.2 Formal Design of Extensible Systems

It has long been recognised (and neglected) that software systems must be designed accordingly if they are to be extensible. Parnas (1978) recalls that:

> The usual programming courses neither mention the need to anticipate change nor do they offer techniques for designing programs in which changes are easy. (Parnas 1978)

If we want to consider in a formal way the design of extensible systems, the situation is even worse. Kramer and Magee (1990), for example, studying the properties of dynamically changing distributed applications, had to develop all their analyses in a textual, informal manner. This is not a general problem since well-established formal methods which can deal with partial correctness and some forms of termination do exist. VDM (Jones 1990) and Z (Spivey 1989) are classical examples but these methods cannot address any form of concurrency. UNITY (Chandy and Misra 1988) overcomes this limitation, although it is not meant for designing open systems, as identified by Fiadeiro and Maibaum (1997). The problem here appears to lie in the fact that these methods were developed without having in mind any of the aforementioned extensibility approaches.

In effect, extensible systems are reactive systems with dynamically varying functionality or structure. In this context, the verification of termination properties becomes less important whereas the possibility of describing concurrent behaviour is paramount given that such systems may be in continuous and simultaneous interaction with many agents in their operational environment. In many cases, termination is not only unnecessary but also forbidden as a violation of a safety property of the system. Moreover, characteristics like naming, which we shall examine in detail later on, are also important in order to deal with reconfigurability and openness. The most prominent formal methods and techniques devoted to capturing these notions are examined below.

### 1.2.1 Process Calculi and Extensibility

Processes and systematic methods of reasoning based on this notion have been around in polished form since the publication of the inspiring paper by Hoare (1978) on the specification language CSP. Later on, Hoare (1985) also developed a collection of proof rules to allow the verification of synchronous concurrent programs. A different theory distinguishing more process non-determinism than CSP was developed by Milner (1980) and called CCS. Milner (1983) also introduced a distinction between synchronous and asynchronous modes of interaction in two different process calculi based on CCS. In addition, he extensively studied notions of equivalence for processes (Milner 1989).

In spite of the widespread use of CSP and CCS, it soon became clear that such languages could not support in direct ways the specification of systems of reconfigurable nature. Moreover, the practice of designing distributed systems showed that more specific modes of interaction between process would be necessary to cover some applications in a realistic manner. Thomsen (1991) proposed two higher-order calculi of processes, where full entities of this kind could be

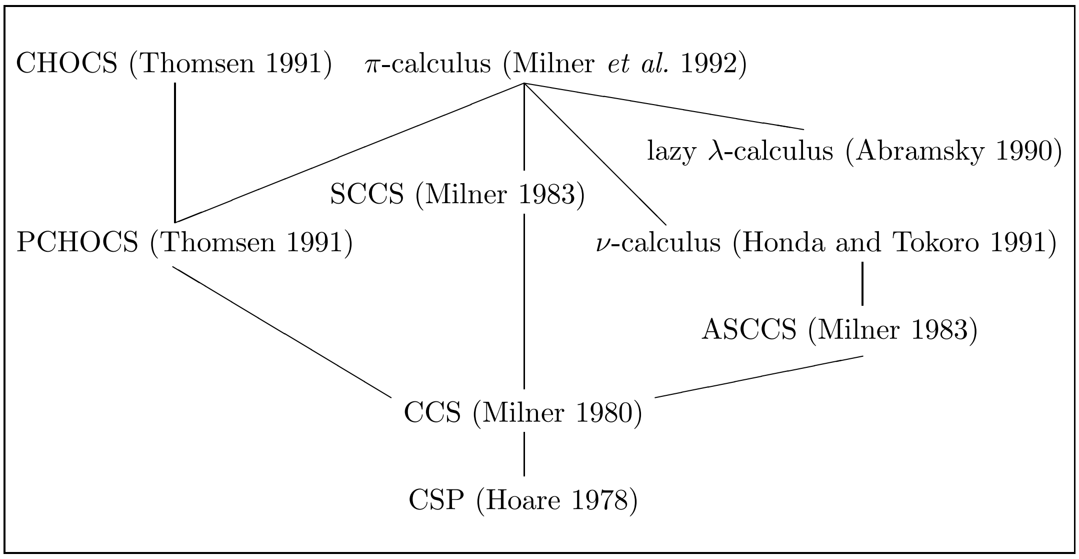


Figure 1.1: Evolution of process design languages.

transmitted as a result of interaction. Honda and Tokoro (1991) opted in the $\nu$-calculus for asynchronous named objects so that only names instead of first class entities could be transmitted in messages. Milner *et al.* (1992) also preferred named entities in the $\pi$-calculus, sticking to a formalism considering only synchronous processes with reconfigurable interconnection topology.

The careful reader may have noticed that these refined process calculi correspond in a way to each of the approaches listed in the previous section, which aim to obtain extensible systems as an outcome of the development process. The $\pi$-calculus of Milner *et al.* (1992) in particular would appear to be the ideal formalism to adopt in designing extensible systems since the object calculus of Honda and Tokoro (1991), the higher-order calculus of Thomsen (1991) and also the lazy $\lambda$-calculus of Abramsky (1990) can all be faithfully embedded in this formalism, as illustrated in Figure 1.1. However, process calculi alone also have their limitations such as the impossibility of specifying and verifying liveness properties, which some real systems must eventually fulfil. This is made worse by the fact that in a stepwise development process some entities may need to be represented as part of a design but will not (and sometimes cannot) be refined into processes in the usual computational sense. These reasons lead us to agree with Tokoro (1993) in that processes appear to be a better abstraction for understanding implementations and the semantics of concurrent programming languages than they are to provide an organised and realistic view of the problem domain. We are thus compelled to look after another kind of formalism.

### 1.2.2 Temporal Logic and Extensibility

Temporal logic has been applied with great success to the specification and verification of software systems since the seminal work of Pnueli (1977). The evolution of this subject area has been constant. Manna and Pnueli (1983) showed how temporal proof systems could be associated to (concurrent) programming languages in a natural way. Barringer (1987) solved the important composability problem, making it possible to rely on the structure of each program in proving temporal properties. Fiadeiro and Maibaum (1992) raised the abstraction level of his work by showing that open concurrent systems could be designed in a modular way in terms of temporal theories. Their results were further extended by Sernadas *et al.* (1995), who developed a temporal logic suitable for object-oriented systems design. Meanwhile, Lamport (1994) and Abadi (1996) have applied the Temporal Logic of Actions in a multitude of domains, treating in particular the development of distributed fault-tolerant systems.

Despite these advances, it is surprising to discover that the design of open reconfigurable systems cannot be directly addressed in detail with any temporal logical system proposed in the literature. In particular, attempting to represent the properties of objects according to the actor model, one easily discovers that a logic which can properly handle object naming as well as presenting a set of connectives with the required meaning is not available. These characteristics are needed in representing some extensible systems accurately.

If compared to process calculi, temporal logics are not suitable for dealing with process or program equivalences but have the fundamental advantage of not committing the whole development process to a fixed abstraction level nor to a fixed abstraction notion, depending of course on how they are defined. The design units may represent programs, processes, theories, objects and others. Temporal specifications in turn may or may not be realisable as executable entities (Abadi *et al.* 1989). Emerson (1983) proposes a helpful classification of temporal logical systems in *exogenous* and *endogenous* depending on whether or not expressions pertaining to the domain of some abstraction notion are covered in the definition of the logical language. Remarkably, all the modal and temporal logics associated to process calculi such as that developed by Milner *et al.* (1993) are of an exogenous nature. Conversely, to achieve enough freedom to apply a temporal logic in describing many problem domains at potentially distinct levels of abstraction, the logic must be an endogenous one. In this thesis, we define and analyse an endogenous temporal logical system.

## 1.3 Aims of the Thesis

To summarise what we have already discussed in this chapter, let us revisit some of the aims of this thesis. Namely, we have hoped to:

- identify what it means for a software system to be extensible;
- identify software development approaches which support extensibility.

We have already provided practical reasons and examples that justify the great importance currently attributed to extensible systems. We have also provided an informal definition of extensibility in terms of possible run time changes and, in addition, a comparison with other related software process notions. These developments allow us to claim that we have characterised extensible software systems. No similar characterisation appears to exist in the literature.

An informal characterisation is not sufficient to support the design of the family of software systems we are aiming at here. In Section 1.2, we examined some classes of theoretical frameworks which are available in the literature and could perhaps be adopted to attempt to accomplish the following two goals:

- to establish theoretical foundations for the design of extensible systems;
- to show that these foundations can be applied in practice to design extensible systems in a rigorous way.

We have argued that the existing formal frameworks cannot be directly applied to design extensible systems. Therefore, by establishing our own formal foundations in terms of a specific temporal logical system, we aim to support their rigorous design and to be able to contrast to each other in an unambiguous manner the characteristics of extensible systems reported in the previous sections, i.e., their design space (Wegner 1987). This study may be useful in deciding which approach to use in representing the distinct situations that arise in practice. The remainder of the thesis deals with these issues.

## 1.4 Outline of the Thesis

Most of the following chapters have the same fixed structure. In the beginning of each chapter, we shall present either a (not necessarily complete) historical retrospective or contextual information which motivates our work. Technical results are subsequently presented and discussed. The last section of each chapter summarises these results and contrasts them to other related work.

Chapter 2 develops our proof-theoretic approach to software development. It begins by describing a rigorous step-by-step way of dealing with software development and its connections with the formal structures of general logics. We examine these connections with the aid of category theory. In the light of this study, we present an incremental axiomatisation of a first-order branching time logic that appears to be an appropriate basis for the design of extensible systems. We examine in detail some characteristics of the logical system such as soundness, completeness and expressiveness.

The subsequent chapter of the thesis shows how to design open reconfigurable systems using a particularisation of the logical system previously defined. We specifically examine in full detail the actor model, proposing an axiomatisation for its features and studying the composition of actor specifications in terms of pushouts in categories of theory presentations. We also show that the model is sufficiently abstract to capture not only distinct modes of interaction but also many approaches to support extensibility. To verify properties of actor systems in a rigorous manner, we adopt a rely-guarantee discipline and prove some meta-logical properties that are helpful in practice.

We continue the investigation on applying our temporal logical system with a study of computational reflection and the design of meta-level architectures in Chapter 4. We show that the assumption of meta-level architectures is reasonable in the design of open reconfigurable systems as formalised in the preceding chapter. We also show that the design of meta-level architectures, despite their apparently circular definition, does not require logics with higher-order features. Moreover, we show that the assumption of an underlying reflective architecture conflicts with systematic software development.

Chapter 5 presents a realistic case study on applying the formal developments of the thesis. We present the specification and verification of a location management architecture in order to illustrate how to design in a rigorous manner software systems that can be extended by mobile components.

The last chapter of the thesis is dedicated to summarising our work and to presenting not only our conclusions but also prospects for future research.

Throughout the thesis, we attempt to use uniform notation and terminology. Indexes pointing to our notational and conceptual definitions are provided at the end of the text, after the bibliography details. Two appendixes are also provided at the end containing the statement and some proofs of properties assumed in the body of the thesis.

# Chapter 2

# Proof Theory and Software Development

Since the seminal work of Floyd (1967), we have hoped to develop an appropriate theory to support rigorous software development. With his inspiring method, Floyd was the first to attempt to ensure in a formal systematic manner that computer programs perform only valid computations, in spite of the practice at that time which was to define merely how each program should compute. His work was centred on associating in a precise way logical assertions to program fragments so as to make possible the proof of partial correctness and termination properties. Admittedly, his method could not scale up to handle the full complexity of real software systems and programming languages.

Another landmark in rigorous software development was the advent of abstract data types (ADTs) as proposed by Liskov and Zilles (1975). ADTs are formal self-contained descriptions of data types and operations in terms of which the whole development process may be understood. They are not meant to stand only for computer programs because the focus of attention in their definition is to describe in a property-oriented relational manner the problem domain, rather than computations, introducing the notion of abstraction in software development. Implementations of ADTs in real programming languages would be obtained at the last stages of the process after a series of refinements.

The studies on the theory of ADTs proved to be very fruitful. Many proof calculi to support verification of properties were proposed by Ehrig and Mahr (1985), Maibaum *et al.* (1985) and by Szałas (1988) for equational, classical and temporal logics, respectively. On the semantic side, algebraic and abstract model theories were developed by Ehrig and Mahr (1985) and by Goguen and Burstall (1992). Perhaps due to this logical diversity, general logics and frameworks were outlined in the work of Meseguer (1990), Meseguer and Martí-Oliet (1995). An approach based on manipulating ADTs using abstraction was established by Ehrich (1982) attempting to make more tractable the process of software development, which in effect could be horizontally and vertically decomposed due to the self-contained character of the manipulated descriptions and to the existence of many abstraction levels, respectively.

Despite their success, it soon became clear that the basic modularisation units of the development process, purely algebraic theories specified by ADTs using some logic, were not well suited to software development in general. To be able to implement ADTs using any imperative language of proven practical value, for instance, it would be necessary for them to embody such notions as state and assignment which could be not be captured explicitly in a purely algebraic manner. Moreover, the assumption that complex systems could always be explained in terms of (possibly divergent) functions defined by algebraic theories put together prevented an appropriate description of concurrent and reactive systems, where mutual interference in intermediate computation steps plays an important role and termination is only a representative of the class of eventuality properties. Not all the proposed logics turned out to be suitable to handle this latter aspect. It has been possible, however, to deal with these issues by fixing the logic as a temporal one (Pnueli 1977) and changing the structuring notion from algebraic to temporal theories (Fiadeiro and Maibaum 1992), even though there is not enough evidence that such an approach would be useful in capturing all the problems of practical interest that may require a computational solution. The same has also been noted considering the notion of process and the respective calculi as reported in our introductory chapter (Milner 1996).

In view of our interest in providing a tractable account for the design of extensible systems, we may infer some important conclusions from the above. The experience with ADTs demonstrates that it is paramount to develop a profound understanding of software development and its underpinning notions to avoid the risk of proposing a theory which cannot be practically used throughout the whole process. For this reason, we shall choose in the sequel modularisation units which are not as concrete as programs nor as inflexible as ADTs, which will be shown adequate by their usefulness in capturing real situations, but cannot be guaranteed to address directly all the problems that may require computational treatment. Understanding their underlying logical system in terms of general logic facilitates the assessment of characteristics like expressibility and composability, as well as to move to a distinct setting in case a real problem is found that cannot be properly treated by the chosen formalism.

The purpose of this chapter is multi-fold. First we outline an approach which we believe provides a better explanation for the process of software development. We try to identify which notions would allow us to treat the process in a formal manner. Next, by providing formal definitions for most of these notions in the context of general logic, we establish our own particular view of both logic and software development, which keeps several similarities with previously proposed frameworks, but nevertheless cannot be fully described in terms of these related works in the way they appear in the literature. We go one step further, applying these notions in the definition of many distinct logical systems, which towards the end of the chapter are unified to form what can be regarded as the original foundational contribution of our work. We critically review in this way most of the background material required to understand the families of systems formalised in the remainder of the thesis.

## 2.1 The Proof-Theoretic Approach

A rigorous and systematic approach to the process of software development has been proposed by Maibaum and Turski (1984). Essentially, the approach relies on the notions of theory and interpretation between theories much in the way that ADT specifications and abstraction are used by Ehrich (1982) to organise the software process. The main difference between these approaches is that the former emphasises the use of syntactical constructions of some logic whereas for ADTs no specific prescription is made. For that reason, the first approach was initially called logical.

The rationale for introducing this distinction, which we entirely agree with, is that it appears more natural to explain software development in terms of theories and their syntactic interconnections than it is using semantical constructions like models and homomorphisms between algebras as studied by Ehrig and Mahr (1985). Take as an example a program implementing a particular specification. From the point of view of a software engineer, it can be made clear how to show in a systematic and direct manner from the source program using the adopted proof calculus that the program satisfies all the constraints posed by the specification and, indeed, implements it. On the other hand, to provide the same kind of assurance using models, their structure must also be formally known *a priori* and only after determining the classes of models of both specification and program is it possible to show an embedding of the latter into the former. Some advocate that the whole process can be justified only on semantical grounds, but by relying merely on models, usually abstract notions without much linguistic structure, we also loose *traceability*, the possibility of identifying precisely how distinct stages of the development process are (linguistically) related.

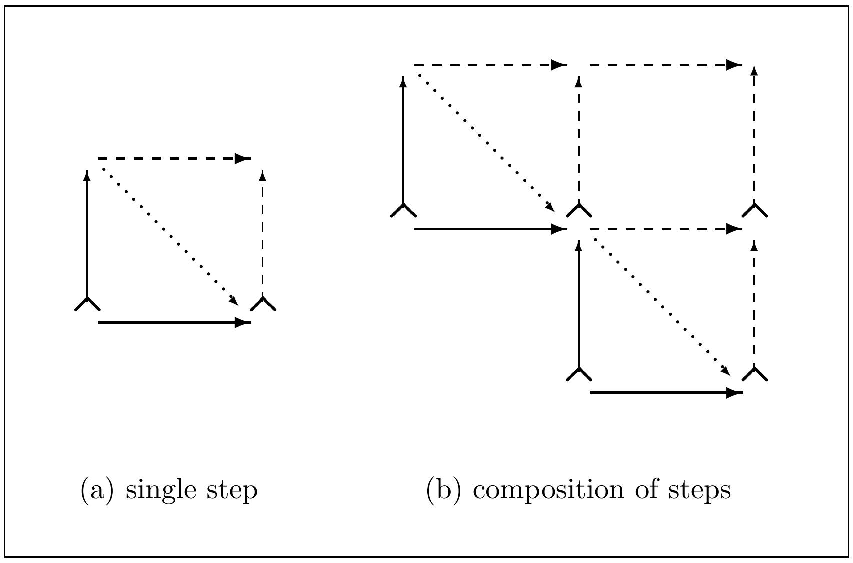


Figure 2.1: Steps of the development process.

The logical approach is rigorously defined in terms of theories and interpretations between theories. As far as an entity can be explained within a full entailment system, endowed with a syntax and a notion of logical consequence, it can be assigned to a theory, its set of consequences. In software development, almost everything can be explained by a theory, from requirements to programs, although such theories are not always formal. The motivation for using theories as modularisation units stems from their explanatory and self-contained character. For instance, every ADT specification determines a theory but the converse is not necessarily true. Having these basic objects at hand, one may want to argue about their relationships and a way to do so is through the use of extensions and translations. As sets, there is a natural notion of extension between theories based on containment. As linguistic constructions, they are equipped with a canonical relation of translation based on the renaming of symbols in their languages. Two particular instances of these are inclusions that are conservative extensions and translations which are interpretations between theories. Conservative extensions prevent the creation of new consequences for the original language within the scope of the extended theory and interpretations preserve the original consequences no matter what their representation is in the new theory. Clearly, none of these notions are necessary, but they are sometimes useful.

The logical approach is systematic in that it prescribes how the stages of the development process should be organised. Starting from an abstract theory, presumably generated by some previously defined specification, an interpretation of this original theory is chosen to serve as a conservative extension of a more concrete resulting theory. Intuitively, extension corresponds to addition of detail while interpretation relates two distinct levels of abstraction. This is illustrated in Figure 2.1. Note that the resulting object does not have to define a program, because many steps may be required before this is achieved; the specification is going to be realised in some other form due to a design decision or it is impossible to produce a program from the current theory. Also note that the order in which extension (denoted by arrows with tails in the figure) and interpretation (represented by single arrows) are computed should be immaterial and once defined it must always yield the same result had the other sequence of operations been chosen. The refinement steps thus defined (represented using dotted arrows) can be composed as operations on theories. To support these features, the meta-theory of the adopted entailment system is required to possess some properties (Maibaum *et al.* 1985).

There is no specific prescription in the logical approach as to which logical system should be used, as soon as it supports the two main activities of rigorous development, design and implementation, in a syntactic manner. Maibaum *et al.* (1984) adopted an infinitary conservative extension of *classical first-order logic*. Actually, the work of Maibaum and Turski (1984) suggested that many systems could be used, one for each stage of the process. Here, since we are only concerned with designs considered in isolation, we may adopt a single logical system, but it is worthwhile mentioning that there is a variety of them to be chosen and each one can make software development more or less painful depending on its features. For instance, it would appear intuitive to regard *propositional intuitionistic logic* as a strong candidate, given its tight connections with the *typed $\lambda$-calculus* via the Curry-Howard isomorphism (Howard 1980), hence with computable functions. However, as already mentioned, software development takes place as a gradual process of decreasing abstraction. It may well be the case that, in the middle of the process, the designer produces a specification intending to describe how a single individual or a community of living entities behave. In such situations, it would be quite restrictive to use an intuitionistic logic. In contrast, choosing proof-calculi without finitary presentation would immediately prevent reasonable automated support.

Concerning the basic building blocks of design, as soon as they define theories, no prescription is made as well. A formal theory may be presented by a finite set of axioms written in a language allowed by the chosen logical system. The original theory may be recovered from these axioms through the

**Theory** PA
**sorts** nat
**constants** $0 : \mathsf{nat}$
**operations** $s : \mathsf{nat} \to \mathsf{nat}; + : \mathsf{nat} \times \mathsf{nat} \to \mathsf{nat}; * : \mathsf{nat} \times \mathsf{nat} \to \mathsf{nat}$
**axioms**

$$\neg(0 = s(x)) \tag{1.1}$$
$$s(x) = s(y) \to x = y \tag{1.2}$$
$$x + 0 = x \tag{1.3}$$
$$x + s(y) = s(x + y) \tag{1.4}$$
$$x * 0 = 0 \tag{1.5}$$
$$x * s(y) = x * y + x \tag{1.6}$$
$$p[x\backslash 0] \wedge (\forall x \cdot p[x] \to p[x\backslash s(x)]) \to \forall x \cdot p[x] \tag{1.7}$$

**End**

Figure 2.2: Classical first-order theory of Peano arithmetic (Kröger 1990).

application of inference rules. For the purpose of software design, the fact that a theory cannot be finitely presented should indicate that either the chosen logical system is not adequate, because it is impossible to represent a problem of interest, or the problem is not to be captured, due to a decision in the design of the formalism. Therefore, it makes sense to restrict our attention to finitely presentable theories and regard only their presentations as specifications. It is important to stress that this requirement is stronger than what is usually understood in logic by the finite axiomatizability of a theory because we require the axiomatisation to be supplied. Interestingly enough, the existence of a finite axiomatisation depends on the chosen logical system. For example, in *first-order logic* the axiomatisation of the *theory of Peano arithmetic* in Figure 2.2 is not finite — (1.7) generates an infinite set of axioms, one for each formula $p$ — nor there is a finite one (Ryll-Nardzwski 1952). Neither of these assertions are true if we consider full *second-order logic* instead.

Initially, all the effort was directed towards characterising how implementation steps could be compartmentalised due to the use of conservative extensions and interpretations between theories. A controversy stated by Diaconnescu *et al.* (1993) concerning the use of an apparent semantic counterpart to the former notion had to be spelled out by Veloso (1992). In essence, the claim was that the software process could be best described in terms of model expansions but, as it turns out, due to the existence of conservative non-expansive extensions, model expansions do not characterise some syntactic constructions of practical interest. Despite these advances, it was only recently that a convincing explanation of horizontal structuring was developed. Fiadeiro and Maibaum (1992) showed that conservative extensions could not be seen as the basic mechanism for composing theories. In point of fact, to achieve *composability* in software development, the possibility of putting theory presentations together to form complex system descriptions, one should be prepared to use creative extensions. The use of such extensions has been identified with the emergence of properties of components when placed in complex configurations (Fiadeiro 1996).

Combining the assumptions of the logical approach and the requirement of using only finite presentations of theories and proof calculi, it seems more sensible to consider the approach above to be proof-theoretic. We stress in this way the fundamental importance of proof-theory as the support upon which specifications, interpretations and their verification, the core objects in rigorous design, are constructed. By this, we are not proposing to abandon model-theory; this does not appear to be appropriate especially in using incomplete logics or trying to achieve higher confidence in a design; nevertheless, we see proof-theoretic constructions as the right objects to deal with in software development.

The approach described so far has been recast in terms of category theory by Fiadeiro and Maibaum (1996). Using this new formulation, let us show as an aside that this approach is useful to clarify the nature of some important properties. The most desirable of these appears to be *compositionality*, which relates horizontal (design) and vertical (implementation) structuring in the development process. Jones (1990) proposes the following characterisation:

> The need is for development methods which have the property that implementations which satisfy specifications of sub-components can be composed so as to satisfy the specification of a system without further proof. A compositional development method permits the verification of a design in terms of the specifications of its sub-programs.
>
> (Jones 1990)

Clearly, compositionality is a relation between the way specifications and programs are composed and verified. Using the terminology of Jones, it means that if we have $S'$ as a specification of a system composed by two specifications $S_1$ and $S_2$ connected through a third one called $S$ and we implement each of them respectively as $P'$, $P_1$, $P_2$ and $P$, we expect the existence of a "unique" way $\eta$ of seeing the program $P'$ as an implementation of $S'$ such that it is a composition of $P_1$ and $P_2$ connected through $P$. This is depicted in Figure 2.3.

The point here is that, in a compositional development process, the original specifications and their structuring are indeed preserved in each implementation step. In categorical terms, this property is captured when we say that there is a functorial relation between the categories of programs and specifications. In the figure, this relation can be represented within the same diagram due to the use of a retrieve functor *Retr* which maps each program into a corresponding specification. The functor plays the role of conservative extensions as explained above and the morphisms interpreting specifications into retrieved programs complete an implementation step. The notion of *satisfaction* of a specification by a program is generalised in this way. The fact that an implementation step is compositional, meaning that $\eta$ is unique up to isomorphism, is automatically ensured whenever *Retr* is a functor (Fiadeiro and Maibaum 1996). All these formal constructions justify the desirable real situation in which it is possible to divide the complex task of verifying that an implementation satisfies a design based on the refinement of its components, as identified by Jones.

Another interesting property called *full abstraction* is often mentioned in the literature. Despite this fact, there does not seem to exist a consensual definition, although some say that this notion is related to the absence of implementation details in each specification:

> A (model-oriented) specification is *biased* on an underlying set of states. The model is biased (with respect to a given set of operations) if there exist different elements of the set of states which cannot be distinguished by any sequence of operations. A model is *sufficiently abstract* providing it can be shown to be free of bias. (Jones 1990)

Moving away from model-oriented specifications as in VDM and their specific notions of state and operation, one may simply say that each biased model contains information which is useless for the particular specification in its current level of abstraction. Specifications in turn are said to be *fully abstract* whenever their models are not biased. Turski and Maibaum have an interesting point of view concerning the description above, which gives us enough motivation to provide a rigorous account of that notion in a similar way to compositionality:

> In full generality, the problem of a specification being without bias, or 'sufficiently abstract' in Jones' terminology, is one that requires a specific context for its resolution. If a specification is considered separately, as an expression of a linguistic level, without a history (the specification for which the current one is an implementation or 'program') and without future (programs that satisfy the current specification) the problem is not very meaningful. (Turski and Maibaum 1987)

Considering this point of view, it appears to be more appropriate to regard full abstraction as the methodological property that distinct programs can be distinguished by some specification. Many distinct ways of refining the same specification may exist and the information it conveys does not need to be totally useful for all purposes, whereas it should be essential for some. A typical example is the systematic addition of concrete details aiming at a specific implementation platform. If a component in a complex configuration is only to read data from a common storage in a shared-variable mode of interaction, none of its operations will change the shared state. Hence, specifications taking this feature into account would appear to be implementation biased if seen in isolation. Considering that the same component is to be implemented in a shared-memory platform, its specification and the adopted refinement method may well be regarded as fully abstract. As in the case of compositionality, we can provide a categorical characterisation of full abstraction as shown in Figure 2.3.

The fact that a refinement method is fully abstract ensures the construction of implementation steps with enough freedom to distinguish through some specification and realisation any pair of distinct programs. Suppose that the refinement $P_1$ of a specification $S_1$ is supported by an interpretation between theories $\iota_1$ and the same happens, respectively, with $S_2$, $P_2$ and $i_2$. We say that the method partially captured by $Retr$ is fully abstract if for any such objects, $[Retr(P_1)]$ and $[Retr(P_2)]$ are equivalent ($\eta$-isomorphic) whenever $P_1$ and $P_2$ are also related in this way by some $\tau$. Note that this is in keeping with the view that a set of possible specifications, determined here by a powerset functor $[\cdot]$, defines the meaning of each program[1]. Seen as above, full abstraction as well as compositionality should be sought in any development method, much in the way that they are in defining programming language semantics (Pnueli 1985b). They are not, however, properties of every method: both are captured when there is a functorial relation between programs and specifications (because functors preserve composition and isomorphisms), but only for some restricted methods relating such categories of objects will they hold.

## 2.2 Logic in General

As illustrated in the previous section, category theory can play a central role in providing a formal and generic account of software development and logic. Instead of stressing the intensional character of collections of objects as in set theory, categories provide an extensional perspective of some problem by focusing mainly on relationships between objects. In what follows, the notion of

[1]The application of the functor $[\cdot]$ to specifications can be defined as $[S] \stackrel{\text{def}}{=} \{S' | S' \to S\}$ and to the morphisms in the category as $[i] \stackrel{\text{def}}{=} [\mathsf{dom}\ i] \to [\mathsf{cod}\ i]$ which is the case if and only if $[\mathsf{dom}\ i] \subseteq [\mathsf{cod}\ i]$. The usual semantics functor is defined as $[\![\cdot]\!] \stackrel{\text{def}}{=} [\cdot] \circ Retr$.

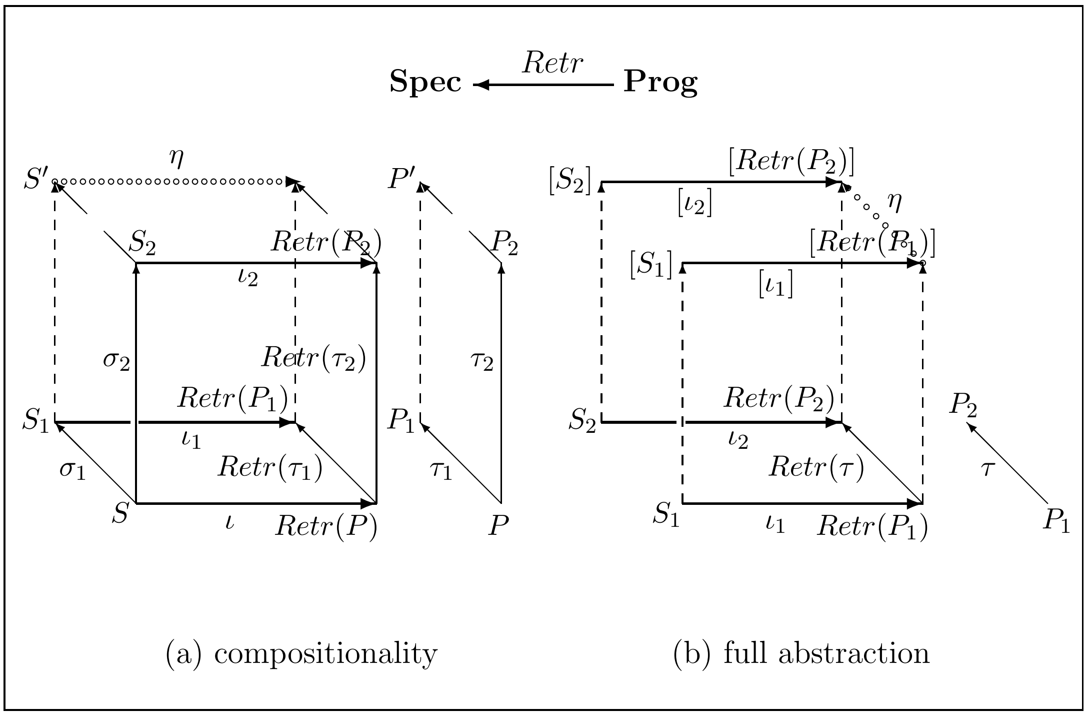


Figure 2.3: Properties of the development process.

category is defined as in the classical textbook of Goldblatt (1979):

**Definition 2.2.1 (Category)** A category $\mathbf{C}$ consists of:

- a collection[2] of entities called *objects*, represented as $\mathsf{obj}\ \mathbf{C}$;
- a collection of entities called *morphisms*, represented as $\mathsf{morph}\ \mathbf{C}$;
- two operations assigning each morphism $f$ of $\mathsf{morph}\ \mathbf{C}$ to objects $\mathsf{dom}\ f$ and $\mathsf{cod}\ f$ in $\mathsf{obj}\ \mathbf{C}$, called the *domain* and *codomain* of $f$, respectively. Each $f$ in $\mathsf{morph}\ \mathbf{C}$ with $\mathsf{dom}\ f = a$ and $\mathsf{cod}\ f = b$ is written as $a \xrightarrow{f} b$;
- an operation $\circ$ called *composition* assigning each two morphisms $f$ and $g$ of $\mathsf{morph}\ \mathbf{C}$ having $\mathsf{dom}\ g = \mathsf{cod}\ f$ to another morphism $(g \circ f)$ in $\mathsf{morph}\ \mathbf{C}$, the *composite* of $f$ and $g$, where $\mathsf{dom}\ (g \circ f) = \mathsf{dom}\ f$ and $\mathsf{cod}\ (g \circ f) = \mathsf{cod}\ g$, such that for every $f$, $g$ and $h$ in $\mathsf{morph}\ \mathbf{C}$ with $\mathsf{dom}\ g = \mathsf{cod}\ f$ and $\mathsf{dom}\ h = \mathsf{cod}\ g$, the *associativity axiom* holds:

  **(ASS)** $h \circ (g \circ f) = (h \circ g) \circ f$;

[2]To make clear our choice of foundational notions concerning set theoretic structures, collection here means a set or a class indistinctly. Hereafter, we shall not worry about such foundational issues.

- an operation assigning a morphism $\mathbf{id}_a$ in $\mathsf{morph}\ \mathbf{C}$, $\mathsf{dom}\ \mathbf{id}_a = \mathsf{cod}\ \mathbf{id}_a = a$, to each object $a$ of $\mathsf{obj}\ \mathbf{C}$, *the identity morphism* of $a$, such that for any $f$ and $g$ in $\mathsf{morph}\ \mathbf{C}$ with $\mathsf{cod}\ f = a = \mathsf{dom}\ g$, the *identity axioms* hold:

  **(ID)** $\mathbf{id}_a \circ f = f$ and $g \circ \mathbf{id}_a = g$.

We have already provided examples of collections of objects, specifications and programs, that may possess enough structure to determine categories, when they are related through morphisms defining interpretations between their theories. In Figure 2.3, we discussed some collective properties of these categories when connected by a *functor*, a morphism of the category of categories (i.e., between categories). As in that case, studying some particular problem, it is almost always the case that one is searching for a *universal property*, characterised by the existence of an object or morphism defined up to isomorphism which enjoys the particular property and is related to each of the other similar members of the category in a unique way. Compositionality, say, was associated to the existence of a unique morphism $\eta$ relating complex specifications to retrieved programs that records the way they were originally composed. As illustrated through that figure, it is sometimes more convenient to study these properties in terms of *diagrams*, the corresponding diagrammatic presentations.

In order to manipulate the basic building blocks of the development process as objects in a category, we rely on the definition of their grammar in terms of *signatures*, usually finite sets of symbols, and on the existence of a relation of consequence between sentences and sets thereof defined in terms of a language allowed by the grammar. In this setting, it is possible to consider theories as objects, interpretations as morphisms and discuss their properties within specific categories of theories. This theory-based view of logic was initially proposed by Fiadeiro and Sernadas (1988) in the form of *$\pi$-institutions*, later revisited by Meseguer (1990) as *entailment systems* and finally generalised by Fiadeiro and Maibaum (1993). In their definition below, we use the notion of sequence and the following notation. Given a set $S$, we use the superscript $+$ in $S^+$ denoting the set of non-empty sequences of $S$-elements and $*$ in $S^* \stackrel{\text{def}}{=} S^+ \cup \{\epsilon\}$ containing the empty sequence $\epsilon$. In addition, we use the subscript $fin$ in $S_{fin}$ representing the set of finite sequences and $\infty$ in $S_\infty$ stands for the respective set of infinite sequences. Of course, these notational conventions may be combined, in which case they have the expected meaning. We also write sequence length as $\mathsf{len}\ S$ and sequence concatenation as $R : T$ or $s : R$, $s \in S$, for sequences $R$, $S$, $T$.

**Definition 2.2.2 (Entailment System)** An *entailment system* is a 5-tuple $\Sigma = (\mathbf{Sig}, \mathcal{L}, \mathcal{E}, \mathcal{G}, \vdash)$ where:

- **Sig** is a category of *signatures*;
- $\mathcal{L}$ is a set of *logical symbols*;
- $\mathcal{E}$ : **Sig** $\rightarrow$ **Set** is a forgetful functor. $\mathcal{E}$ associates each signature $\Delta$ in $\mathsf{obj}$ **Sig** to its set of *extra-logical symbols* such that $\mathcal{E}(\Delta) \cap \mathcal{L} = \{\,\}$;
- $\mathcal{G}$ : **Sig** $\rightarrow$ **Set** is a functor defining a *language grammar.* $\mathcal{G}$ associates to each signature $\Delta$ in $\mathsf{obj}$ **Sig** a set of *legal sentences* $\mathcal{G}(\Delta) \subseteq S^{+}$ where $S \overset{\text{def}}{=} \mathcal{E}(\Delta) \cup \mathcal{L}$. For $p \in \mathcal{G}(\Delta)$, $p \equiv s_1 \ldots s_n$, we extend the definition of $\mathcal{E}$ as follows: $\mathcal{E}(p) = \{s_i | s_i \in \mathcal{E}(\Delta)\}$;
- $\vdash$ is a function associating to each $\Delta$ in $\mathsf{obj}$ **Sig** a binary *entailment relation* $\vdash_{\Delta} \subseteq \mathcal{P}(\mathcal{G}(\Delta)) \times \mathcal{G}(\Delta)$. $\Sigma$ is *fully strongly (weakly) structural* if and only if for any $\Psi_1 \cup \Psi_2 \cup \{p, q\} \subseteq \mathcal{G}(\Delta)$, the following conditions are satisfied:

  1. *reflexivity:* $\Psi_1 \vdash_{\Delta} p$ for every $p \in \Psi_1$;
  2. *monotonicity:* if $\Psi_1 \vdash_{\Delta} p$ and $\Psi_1 \subseteq \Psi_2$ then $\Psi_2 \vdash_{\Delta} p$;
  3. *transitivity:* if $\Psi_1 \vdash_{\Delta} p$ for every $p \in \Psi_2$ and $\Psi_1 \cup \Psi_2 \vdash_{\Delta} q$, $\Psi_1 \vdash_{\Delta} q$;
  4. *strong (weak) structurality:* for every $\tau : \Delta \rightarrow \Delta'$, if $\Psi_1 \vdash_{\Delta} p$, there is an empty (finite) $\Psi'_{\tau} \subseteq \mathcal{G}(\Delta')$ such that $\mathcal{G}(\tau)(\Psi_1) \cup \Psi'_{\tau} \vdash_{\Delta'} \tau^{\#}(p)$, where $\tau^{\#}(p)$ is defined by pointwise application of $\tau$: for $1 \leq i \leq \mathsf{len}\ p$, $\tau^{\#}(p_i) = p_i$ if $p_i \in \mathcal{L}$ or $\tau^{\#}(p_i) = \tau(p_i)$ otherwise. $\square$

Roughly speaking, an entailment system supports the manipulation of a family of theories based on three components: a category of signatures; a family of languages endowed with a common grammar and a classification of their ground symbols; and a family of entailment or consequence relations, one for each signature. Signatures in general have some additional structure, which can be forgotten through the functor $\mathcal{E}$ yielding a set of extra-logical symbols. The language grammar can only be defined in terms of these symbols together with logical symbols, such as connectives, variables and others in $\mathcal{L}$. This requirement cannot be found in (Fiadeiro and Sernadas 1988, Meseguer 1990, Fiadeiro and Maibaum 1993) and is introduced here due to the assumption that each entailment system has a closed vocabulary of symbols. That is, although distinct extra-logical symbols may appear in each signature, despite the fact that the choice of their names is immaterial because they can be renamed by signature morphisms, this assumption rules out the introduction of new logical constants as the system is used. We would not be able to regard our approach to logic as formal if new symbols and notation could be introduced at will, especially because it would be impossible to develop meta-logical results such as the deduction theorem which depend on performing inductive arguments over the languages of the system. Providing the required symbols explicitly also facilitates extending the definition of entailment systems to deal with theories as described below.

The properties of full entailment relations are usually found in any kind of logical consequence. Reflexivity says that everything assumed is entailed. Monotonicity guarantees that with more assumptions we can never conclude less properties. Actually, we have favoured monotonicity in lieu of compactness, as proposed by Fiadeiro and Sernadas (1988), since some of the temporal logics we shall study fail to guarantee that any property entailed by a set of assumptions is also entailed by a finite subset thereof. Transitivity, sometimes confusingly called cut, captures the fact that using conclusions as assumptions does not allow us to conclude more properties. Note that entailment relations do not capture the relation of derivability between sets of sentences and single sentences when they are manipulated by inference rules of a proof calculus. For *classical first-order logic*, say, $\vdash_\Delta$ captures the validities over the signature $\Delta$, which are independent from the proof calculus adopted.

One notion that entailment relations can capture is the possibility of translating a validity over a signature into another one belonging to the language of a different signature. This is useful when it is necessary to proceed in a derivation within the context of a distinct presentation. For that effect, Meseguer (1990) proposed $\vdash$-translation: that the translation of an entailed sentence is entailed by the translation of the set of sentences which supported the original relationship, i.e., strong structurality. It turns out that, for some logical systems of practical interest which we shall study in the next chapter, this condition is too strong. In a sense, the target entailment may be too weak to support the original one as such. That is why the existence of a finite set of sentences $\Psi'_\tau$ is required in (4) so that, adjoined to the translation of the original set, ensures the entailment of the translated sentence (Fiadeiro and Maibaum 1993). In most cases, $\vdash$-translation is enough and can be recovered by putting $\Psi'_\tau = \{\,\}$.

Theories are defined as follows. Given an entailment system (**Sig**, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{G}$, $\vdash$) and a set of sentences $\Psi \subseteq \mathcal{G}(\Delta)$ over $\Delta$ in $\mathsf{obj}$ **Sig**, the set of *theorems* $Th^{\vdash}_{\Delta}(\Psi) \stackrel{\text{def}}{=} \{p \in \mathcal{G}(\Delta) | \Psi \vdash_\Delta p\}$ is called the *theory* of $\Psi$ over $\Delta$. It is thus the *closure* of $\Psi$ under the binary relation $\vdash_\Delta$. Theories and set inclusion determine a category **Th**.

Using the definition of theory, we can lift a category of signatures **Sig** to a category of logical theories **Theo** as follows. For each $\Delta$ in $\mathsf{obj}$ **Sig**, $Th_\Delta(\Psi)$ in $\mathsf{obj}$ **Theo** if and only if $\Psi \subseteq \{p \in \mathcal{G}(\Delta) | \{\,\} \vdash_\Delta p\}$. Moreover, for each **Sig**-

morphism $\sigma : \Delta \rightarrow \Delta'$ and each $\{Th_\Delta(\Psi), Th_{\Delta'}(\Psi')\}$ contained in obj **Theo**, there is a morphism $\tau : Th_\Delta(\Psi) \rightarrow Th_{\Delta'}(\Psi')$ in morph **Theo** if and only if $\{\sigma^\#(p) \in \mathcal{G}(\Delta') | p \in Th_\Delta(\Psi)\} - \Psi'_\sigma \subseteq Th_{\Delta'}(\Psi')$, where $\sigma^\#$ and $\Psi'_\sigma$ are as in Definition 2.2.2. We say that a theory morphism $\tau : Th_\Delta(\Psi) \rightarrow Th_{\Delta'}(\Psi')$ having $\sigma : \Delta \rightarrow \Delta'$ as underlying signature morphism is an *interpretation between theories* whenever $\{\sigma^\#(p) \in \mathcal{G}(\Delta') | p \in Th_\Delta(\Psi)\} \subseteq Th_{\Delta'}(\Psi')$. Furthermore, $\tau$ is said to be *faithful*[3] if and only if, for every $p \in \mathcal{G}(\Delta)$, $\Psi' \vdash_{\Delta'} \sigma^\#(p)$ if and only if $\Psi \vdash_\Delta p$. We can now define what is meant by a *theory presentation*:

**Definition 2.2.3 (Theory Presentation)** Given an entailment system (**Sig**, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{G}$, $\vdash$), a *theory presentation* is a pair $\Phi = (\Delta, \Psi)$ where:

- $\Delta$ in obj **Sig** is a signature;
- $\Psi \subseteq \mathcal{G}(\Delta)$ is a finite set of *extra-logical axioms*. □

As we have already hinted, a weaker notion is that of a *finitely axiomatizable* theory: $Th_\Delta(\Psi)$ is finitely axiomatizable if and only if there is a theory presentation $(\Delta, \Psi')$ such that $\Psi' \vdash_\Delta p$ whenever $p \in Th_\Delta(\Psi)$ and only then. The lifting of **Sig** to **Theo** naturally extends to categories of presentations **Pres** and finitely axiomatizable theories **FinAx** by requiring respectively that $\Psi$ be finite or $Th_\Delta(\Psi)$ be finitely axiomatizable for each $\Psi \subseteq \mathcal{G}(\Delta)$. In practice, we often work with the respective sub-categories of **Th**.

A formal account to logic would not be accurate without treating the notions of model and satisfaction. The theory of *institutions* proposed by Goguen and Burstall (1992) can be used to deal with these semantic notions in an abstract manner:

**Definition 2.2.4 (Institution)** An *institution* is a 4-tuple (**Sig**, $\mathcal{G}$, $Mod$, $\models$) where:

- **Sig** is a category of signatures;
- $\mathcal{G}$ : **Sig** $\rightarrow$ **Set** is a functor defining the language grammar;
- $Mod$ : **Sig** $\rightarrow$ **CAT**$^{op}$ is a functor associating to each $\Delta$ in obj **Sig** a category $Mod(\Delta) = \mathbf{Mod}_\Delta$ of *models* of $\Delta$;
- $\models$ is a function associating to each $\Delta$ in obj **Sig** a binary *satisfaction relation* $\models_\Delta \subseteq$ obj $\mathbf{Mod}_\Delta \times \mathcal{G}(\Delta)$. For any $\tau : \Delta \rightarrow \Delta'$, $p \in \mathcal{G}(\Delta)$ and $\theta'$ in obj $\mathbf{Mod}_{\Delta'}$, $\theta' \models_{\Delta'} \tau^\#(p)$ iff $Mod(\tau)(\theta') \models_\Delta p$. □

[3]If $\tau : (\Delta, \Psi) \rightarrow (\Delta, \Psi')$ is a faithful morphism, then it captures the conservative extension $Th_\Delta(\Psi) \subseteq Th_\Delta(\Psi')$.

In institutions, the category of signatures and the grammar functor are similar to the components of an entailment system, but the requirement of vocabulary closure is absent because this neither can be a general model-theoretic property nor does it appear in the original definition of institution. Actually, it would falsify any Upward Skolem-Löwenheim theorem (see van Dalen (1994) for an example). The functor $Mod$ associates signatures to categories of models belonging to $\mathsf{obj}\ \mathbf{CAT}^{op}$, the dual to the category of categories $\mathbf{CAT}$ with all morphisms reversed. The function $\models$ is analogous to $\vdash$ and each satisfaction relation respects the semantic counterpart to strong-structurality which requires that truth be invariant under change of notation, the *satisfaction condition.* Whenever this semantic condition is too strong to be obtained, e.g. in weakly structural entailment systems, we shall indicate how it can be approximated.

The definition of the functor $Mod$ and the function $\models$ can be extended to the categories of theories induced by the signatures of a specific full entailment system. For instance, $Mod : \mathbf{Th} \to \mathbf{Cat}^{op}$ associates each theory $Th_\Delta(\Psi)$ to a category of models $\mathbf{Mod}_{Th_\Delta(\Psi)}$, where $\theta$ in $\mathsf{obj}\ \mathbf{Mod}_{Th_\Delta(\Psi)}$ if and only if we have $\theta \models_\Delta p$ for every $p \in Th_\Delta(\Psi)$. The semantic consequence relation $\models_\Delta \subseteq \mathcal{P}(\mathcal{G}(\Delta)) \times \mathcal{G}(\Delta)$ is defined as $\Psi \models_\Delta p$ if and only if $\theta \models_\Delta p$ whenever $\theta \models_\Delta q$ for every $q \in \Psi$, for every $\theta$ in $\mathsf{obj}\ \mathbf{Mod}_\Delta$. For a fixed $\theta$, we write $\Psi \models^\theta_\Delta p$. This relation generates a semantic notion of theory $Th^{\models}_\Delta(\Psi)$ complementing $Th^{\vdash}_\Delta(\Psi)$.

A *logic* is defined by putting together a full entailment system and an institution so that they share the same category of signatures, but the closure of the syntactic grammar is semantically forgotten, obeying the following:

**Definition 2.2.5 (Logic)** A *logic* is a 9-tuple ($\mathbf{Sig}$, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{G}^{\vdash}$, $\vdash$, $\mathcal{G}^{\models}$, $Mod$, $\models$, $\alpha$) where:

- ($\mathbf{Sig}$, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{G}^{\vdash}$, $\vdash$) is a full entailment system;
- ($\mathbf{Sig}$, $\mathcal{G}^{\models}$, $Mod$, $\models$) is an institution;
- $\alpha : \mathcal{G}^{\vdash} \Rightarrow \mathcal{G}^{\models}$ is a natural isomorphism;

such that the *soundness condition* holds: for any $\Delta \in \mathsf{obj}\ \mathbf{Sig}$ and each $\Psi \cup \{p\} \subseteq \mathcal{G}^{\vdash}(\Delta)$, $\Psi \vdash_\Delta p$ implies $\{\alpha_\Delta(q) | q \in \Psi\} \models_\Delta \alpha_\Delta(p)$. If, in addition, the converse of this condition holds, then the logic is said to be *complete.* □

We say that an entailment relation is *compact* if and only if for every $\Delta$, $\Psi \vdash_\Delta p$ implies the existence of a finite $\Psi' \subseteq \Psi$ such that $\Psi' \vdash_\Delta p$. The same applies to semantic consequence relations. Sometimes, due to the failure of compactness, completeness as defined above cannot be obtained. In this sense, our definition is of a strong notion occasionally called *adequacy.* It may be useful to study *weak* and *medium* completeness notions where $\Psi$ is considered to be always empty and finite, respectively. It also makes sense to talk about *relative* soundness or completeness notions, where the entailment and satisfaction relations of distinct logics are related according to the conditions in the definition above. The differences between $\mathcal{G}_{\vdash}$ and $\mathcal{G}_{\models}$ as well as between $Th^{\vdash}_{\Delta}$ and $Th^{\models}_{\Delta}$ for each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}$ are normally ignored for the sake of simplicity.

As discussed in the previous section, we consider the notion of *proof calculus* to be the basis upon which our approach to software development is defined. Each proof calculus provides a systematic method, defined in terms of the notions of axiom schema and inference rule, for determining whether or not a single sentence is a consequence of a set of sentences. It also appears to be reasonable to say that, when an entailment system is associated to a proof calculus, this last structure generates each theory and supports their manipulation, in that the calculus provides rigorous tools for classifying theory morphisms and finding derived properties. Some attempts to capture the notion of proof calculus in generic form have already appeared in the literature. Meseguer (1990), to abstract away the structure of each derivation through category theory, used a generalised formal construction called multi-category. A distinct approach was adopted by Harper *et al.* (1994), who studied the representation of proof calculi using judgement rules of a particular *type theory.* Here, since we do not want to commit ourselves to any additional formal apparatus, a set-theoretic definition is proposed below:

**Definition 2.2.6 (Proof calculus)** A *proof-calculus* is an 8-tuple ($\mathbf{Sig}$, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{G}$, $\vdash$, $Ax$, $\Vdash$, $Pr$) where:

- ($\mathbf{Sig}$, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{G}$, $\vdash$) is an entailment system;
- $Ax : \mathbf{Sig} \to \mathbf{Set}$ is a functor assigning each signature $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}$ to a set of *logical axioms* $Ax(\Delta) \subseteq \mathcal{G}(\Delta)$ such that $p \in Ax(\Delta)$ implies $\{\,\} \vdash_{\Delta} p$. $Ax(\Delta)$ is generated by a finite set of *axiom schemas* written in terms of *schematic variables* ranging over $\mathcal{G}(\Delta)$ and logical symbols in $\mathcal{L}$;
- $\Vdash$ is a function associating each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}$ to a binary *derivability relation* $\Vdash_{\Delta} \subseteq \mathcal{P}^{+}(\mathcal{P}(\mathcal{G}(\Delta)) \times \mathcal{G}(\Delta)) \times \mathcal{G}(\Delta)$ such that $\Psi \vdash p$ whenever $(\chi, p) \in \Vdash_{\Delta}$ and $\Psi = \{q | \exists \delta \cdot (\delta, q) \in \chi\}$. Each $p$ is a *conclusion*, $\Psi$ and $\delta$ are sets of *premises* and of *assumptions to be discharged* respectively. $\Vdash_{\Delta}$ is generated by the application of a finite set of *inference rules* written in terms of schematic variables and logical symbols;

- $Pr$ is a function associating each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}$ and $\Psi \cup \{p\} \subseteq \mathcal{G}(\Delta)$ to a set $Pr_\Delta(\Psi, p)$ of tree-structured *derivations* of a conclusion $p$ from a set of *assumptions* or *hypotheses* $\Psi$. $Pr_\Delta(\Psi, p)$ is the smallest set of derivations organised in *proof steps* according to the following inductive scheme:

    1. if $p \in \Psi$ then $(\{\,\}, p) \in Pr_\Delta(\{p\}, p)$ (assertion of an assumption);
    2. if $p \in Ax(\Delta)$ then $(\{\,\}, p) \in Pr_\Delta(\{\,\}, p)$ (use of an axiom schema);
    3. if $\chi = \{(\delta_i, p_i) | \delta_i \cup \{p_i\} \subseteq \mathcal{G}(\Delta)\}$ (a set of derivation contexts), $D = \{(d_i, p_i) \in Pr_\Delta(\psi_i \cup \delta_i, p_i) | \psi_i \subseteq \Psi, \exists c \in \chi \cdot c = (\delta_i, p_i)\}$ (a set of derivations) and $\chi \Vdash_\Delta p$ then $(D, p) \in Pr_\Delta(\Psi, p)$ (application of an inference rule)[4];

    such that the *faithfulness condition* is postulated: $Pr_\Delta(\Psi, p) \neq \{\,\}$ and $\forall (D, p) \in Pr_\Delta(\Psi, p) \cdot (D = \{\,\} \rightarrow \Psi = \{\,\})$ iff $\Psi \vdash_\Delta p$. $\square$

A *proof* of $p$ over a presentation $(\Delta, \Psi)$ is a derivation of $p$ with $\Psi$ as the set of hypotheses. We say that $p$ is *derivable* from $\Psi$ in this case. A *generic proof* of $p$ is a derivation of $p$ with the empty set of hypotheses, in which case $p$ is said to be *provable*. A proof calculus is said to be *formal* only if derivations, which have finite length, are composed solely by the application of inference rules taking a finite number of premises. Otherwise, the calculus is considered to be *semi-formal*, differing from informal structures just because of its rigorous, though not finitary, definition. An example of a semi-formal calculus is that of *$\omega$-logic*, defined by Chang and Keisler (1977) as an extension of classical first-order natural deduction with an infinitary inference rule which takes an infinite number of instances of a formula as premises, one for each natural number, and allows the conclusion of its universal generalisation as a quantified sentence.

The definition of $Pr$ deserves further attention. A set of application examples is provided in Figure 2.4 to show that $Pr$ is general enough to capture the usual proof calculus styles. As stated above, $Pr_\Delta(\Psi, p)$ is a set of derivations of a conclusion from a set of assumptions. We use this set to define a family of entailment relations by postulating a faithfulness condition. Note that we disregard single assertions of assumptions as generating entailments to prevent them from always being reflective. We could have also assumed the existence of a set of logical labels and considered labelled sentences and proof steps. That

[4]It is worthwhile mentioning that we consider the different ways of dealing with assumptions using inference rules as the only essential distinction between the usual proof calculi styles: while *Hilbert-style* calculi do not allow us to discharge assumptions using inference rules and prioritise in this way axiom schemas, *natural deduction* rules discharge assumptions explicitly and *sequent calculi* rules internalise this treatment.

*Hilbert style:*

1. $s_2^a : p \land (q \lor r) \leftrightarrow (p \land q) \lor (p \land r)$ **DIST-AO** (see Appendix-I)
2. $(D_1^a, s_1^a) : p \land (q \lor r) \rightarrow (p \land q) \lor (p \land r)$ **IFF-E** 1 (see Appendix-I)

This annotated generic proof is justified by the fact that $s_2^a \in Ax(\Delta)$ (first case in Definition 2.2.6) and that $(D_1^a, s_1^a) \in Pr_\Delta(\{\,\}, s_1^a)$ (second case in the definition), allowed by the schema and rule stated in Appendix I. Note that $D_1^a = \{(\{\,\}, s_2^a)\}$.

*Natural deduction:*

$$\dfrac{\dfrac{\dfrac{s_{4.1}^b : [p \land (q \lor r)]}{(D_{3.1}^b, s_{3.1}^b) : q \lor r}\,{\scriptstyle \land\mathcal{E}} \qquad \dfrac{\dfrac{\dfrac{s_{6.1}^b : [p \land (q \lor r)]}{(D_{5.1}^b, s_{5.1}^b) : p}\,{\scriptstyle \land\mathcal{E}} \quad s_{5.2}^b : [q]}{(D_{4.1}^b, s_{4.2}^b) : p \land q}\,{\scriptstyle \land\mathcal{I}}}{(D_{3.2}^b, s_{3.2}^b) : (p \land q) \lor (p \land r)}\,{\scriptstyle \lor\mathcal{I}} \qquad (D_{3.3}^b, s_{3.3}^b)}{(D_2^b, s_2^b) : (p \land q) \lor (p \land r)}\,{\scriptstyle \lor\mathcal{E}}}{(D_1^b, s_1^b) : p \land (q \lor r) \rightarrow (p \land q) \lor (p \land r)}\,{\scriptstyle \rightarrow\mathcal{I}}$$

where $(D_{3.3}^b, s_{3.3}^b) \cong (D_{3.2}^b, s_{3.2}^b)$.
The proof above is justified in a similar way to the Hilbert-style case. The application of inference rules, which generate each $D_{i.j}$, is permitted by the natural deduction rules. The novelty in this case is the discharge of assumptions. For example, note that $(D_{3.2}^b, s_{3.2}^b) \in Pr_\Delta(\{s_{6.1}^b, s_{5.2}^b\}, s_{3.2}^b)$, $(D_{3.3}^b, s_{3.3}^b) \in Pr_\Delta(\{s_{6.2}^b, s_{5.4}^b\}, s_{3.3}^b)$. Therefore, due to the $\lor\mathcal{E}$ rule, $\{(\{\}, s_{3.1}^b), (\{s_{5.2}^b\}, s_{3.2}^b), (\{s_{5.4}^b\}, s_{3.3}^b)\} \Vdash_\Delta s_2^b$ and then $(D_2^b, s_2^b) \in Pr_\Delta(\{s_{4.1}^b\}, s_2^b)$, because $s_{4.1}^b \cong s_{6.1}^b \cong s_{6.2}^b$.

*Sequent calculus:* We assume the existence of a logical symbol $\Rightarrow$ in each sentence.

$$\dfrac{\dfrac{\dfrac{\dfrac{s_{5.1}^c : p \Rightarrow p}{(D_{4.1}^c, s_{4.1}^c) : p \Rightarrow p, q \lor r}\,{\scriptstyle \mathcal{WR}}}{(D_{3.1}^c, s_{3.1}^c) : p \land (q \lor r) \Rightarrow p, q \lor r}\,{\scriptstyle \land\mathcal{L}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{s_{7.1}^c : p \Rightarrow p}{(D_{6.1}^c, s_{6.1}^c) : p, q \Rightarrow p}\,{\scriptstyle \mathcal{WL}} \quad (D_{6.2}^c, s_{7.2}^c)}{(D_{5.1}^c, s_{5.2}^c) : p, q \Rightarrow (p \land q)}\,{\scriptstyle \land\mathcal{R}}}{(D_{4.2}^c, s_{4.2}^c) : p, q \Rightarrow (p \land q) \lor (p \land r)}\,{\scriptstyle \lor\mathcal{R}} \qquad (D_{4.3}^c, s_{4.3}^c)}{(D_{3.2}^c, s_{3.2}^c) : p, q \lor r \Rightarrow (p \land q) \lor (p \land r)}\,{\scriptstyle \lor\mathcal{L}}}{(D_2^c, s_2^c) : p \land (q \lor r) \Rightarrow (p \land q) \lor (p \land r)}\,{\scriptstyle \mathcal{CUT}}}{(D_1^c, s_1^c) :\Rightarrow p \land (q \lor r) \rightarrow (p \land q) \lor (p \land r)}\,{\scriptstyle \rightarrow\mathcal{R}}$$

where $(D_{4.3}^c, s_{4.3}^c) \cong (D_{4.2}^c, s_{4.2}^c)$ and $(D_{6.2}^c, s_{7.2}^c) \cong (D_{6.1}^c, s_{7.1}^c)$.
Disregarding the treatment of assumptions, this example is similar to the preceding one. The terminal sentences in sub-derivations, e.g. $s_{5.1}$ and $s_{7.1}$, belong to $Ax(\Delta)$. These are generated by a standard schema in sequent calculi, $p \Rightarrow p$, $p \in \mathcal{G}(\Delta)$.

Figure 2.4: Example of distinct proof calculi styles.

would be to record the inference rule justifying each step in a derivation, which is sometimes useful when there are many different ways to derive a conclusion from the same set of premises and also to provide rigorous control over discharging of assumptions in natural deduction like calculi. We prefer to avoid this additional complexity for the sake of simplicity. Furthermore, it is possible to relax the faithfulness condition to introduce new soundness and completeness relationships as proposed by Avron (1991), this time between entailment system and proof calculus. We consider the equivalence above to be essential because the entailment and satisfaction relations of a logic are already related according to these conditions. Giunchiglia and Serafini (1994) study derivability relations where premises and conclusion belong to distinct logical systems. An extension of our definitions towards this direction is clearly subject for further work.

It is also important to mention that, for a given signature $\Delta$, $\Vdash_\Delta$ does not inherit the properties of the underlying entailment relation. It only captures particular applications of inference rules of the proof calculus, for which reflexivity, say, would mean that for each premise there is a rule which allows us to repeat such a sentence in the subsequent proof step, a requirement which is not acceptable in general[5]. Seen as a binary relation, $Pr_\Delta$ inherits all the properties of $\vdash_\Delta$. By abuse of notation, $p \in Ax(\Delta)$ is normally written as $\Vdash_\Delta p$. Moreover, we write $\Psi \Vdash_\Delta p$ or $\Vdash_{(\Delta,\Psi)} p$ whenever $Pr_\Delta(\Psi, p)$ is not empty.

In the remainder of the chapter, we will be interested in providing presentations for proof calculi of some interesting *logical systems*:

**Definition 2.2.7 (Logical System)** A *logical system* is a 12-tuple ($\mathbf{Sig}$, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{G}^\vdash$, $\vdash$, $\mathcal{G}^\models$, $Mod$, $\models$, $\alpha$, $Ax$, $\Vdash$, $Pr$) where:

- ($\mathbf{Sig}$, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{G}^\vdash$, $\vdash$, $\mathcal{G}^\models$, $Mod$, $\models$, $\alpha$) is a logic;
- ($\mathbf{Sig}$, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{G}^\vdash$, $\vdash$, $Ax$, $\Vdash$, $Pr$) is a proof calculus; $\square$

A logical system is considered to be *effective* if provability is *decidable* for the underlying proof calculus, meaning that it is possible to write an algorithm which decides whether or not there is a generic proof for each sentence.

An inference rule $\Psi \Vdash_\Delta P$, $\Psi$ and $P$ written in terms of schematic variables and logical symbols, is considered to be derivable in a logical system $\mathcal{S}$ if and only if for every pair of instances ($\psi$, $p$) of ($\Psi$, $P$), $Pr_\Delta(\psi, p)$ is not empty. The same rule is said to be admissible in $\mathcal{S}$ if and only if $\models_\Delta p$ whenever $\models_\Delta \psi$. These definitions are standard in the literature (Rybakov 1997). Clearly, derivable rules are admissible by definition. Derived rules make the application of a proof

[5]But see Friedman and Sheard (1995) for a "proof calculus" with such rule.

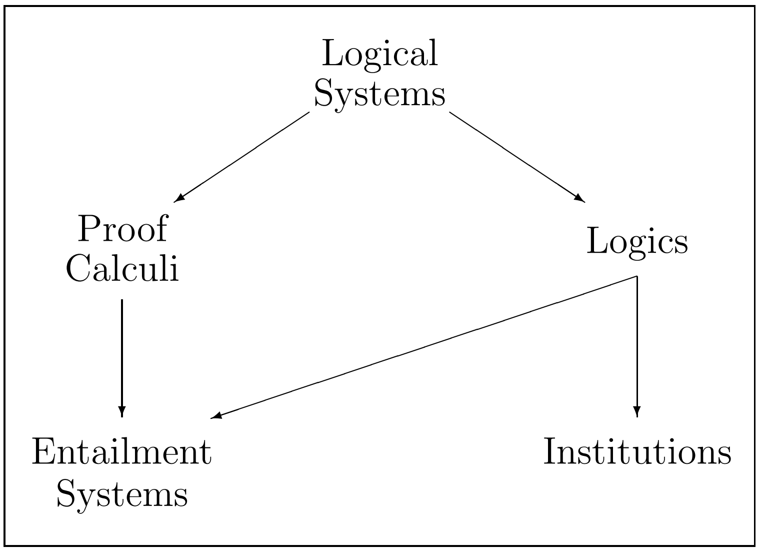


Figure 2.5: A taxonomy of logical structures.

calculus easier in practice while the incorporation of an admissible rule results in a more powerful proof theory. We deal with both kinds of rules in the sequel.

At this point, we should remind the reader that we are not attempting to propose an original formulation of general logic. Rather, we have made an effort to establish practical foundations in order to support a rigorous investigation of many different logical systems which are to be introduced. Providing definitions for general logical structures interconnected as depicted in Figure 2.5 allows us to study meta-logical properties in a logic independent manner, to determine to what extent — based on what assumptions — general properties hold and to transport these results elsewhere whenever possible and necessary.

## 2.3 Classical Propositional Logic

From this section onwards, our purpose will be to define a logical system to support the design of extensible systems. We shall define some distinct entailment systems in terms of their respective Hilbert-style proof calculi and examine how they are connected to each other and used in isolation, postponing the definition of the associated model-theoretic notions until the final sections, after having defined the whole proof-theoretic structure. We begin by looking at *classical propositional logic*. Since this logic is quite well-understood (a comprehensive study is developed by van Dalen (1994)), we take advantage of this fact to illustrate how a proof-theoretic approach leads us to define an entailment system.

**Definition 2.3.1 (Classical Propositional Logic)** The entailment system of *classical propositional logic*, $CPL$ for short, is defined as follows:

- $\mathbf{Sig}^{CPL} \cong \mathbf{FinSet}$ (i.e., $\mathbf{Sig}^{CPL}$ is isomorphic to the category of finite sets);
- $\mathcal{L}^{CPL} \stackrel{\text{def}}{=} \{\neg, \rightarrow, (, )\}$;
- $\mathcal{E}^{CPL} \cong \mathbf{id}_{\mathbf{Sig}^{CPL}}$. For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{CPL}$, each element of $\mathcal{E}^{CPL}(\Delta)$ is called a *proposition symbol*;
- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{CPL}$, $\mathcal{G}^{CPL}(\Delta)$ is a set of *propositions* defined by $P^{CPL}$ as follows, provided that $p \in \mathcal{E}^{CPL}(\Delta)$:

  $$P^{CPL} ::= p \mid \neg P^{CPL} \mid (P^{CPL} \rightarrow P^{CPL})$$

  We shall ignore superfluous parentheses in propositions and adopt the usual precedence conventions. We also use the following abbreviations for each $\{p, q\} \subseteq \mathcal{G}^{CPL}(\Delta)$:

  **(D1-$\top$)** $\top \stackrel{\text{def}}{=} p \rightarrow p$;

  **(D2-$\bot$)** $\bot \stackrel{\text{def}}{=} \neg\top$;

  **(D3-OR)** $p \vee q \stackrel{\text{def}}{=} (\neg p \rightarrow q)$;

  **(D4-AND)** $p \wedge q \stackrel{\text{def}}{=} \neg(p \rightarrow \neg q)$;

  **(D5-IFF)** $p \leftrightarrow q \stackrel{\text{def}}{=} (p \rightarrow q) \wedge (q \rightarrow p)$ [or $\neg((p \rightarrow q) \rightarrow \neg(q \rightarrow p))$];

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{CPL}$, the entailment relation $\vdash^{CPL}_{\Delta}$ is generated by the following proof calculus, provided that $\{p, q, r\} \subseteq \mathcal{G}^{CPL}(\Delta)$[6]:

  **(A1-I)** $\Vdash^{CPL}_{\Delta} p \rightarrow (q \rightarrow p)$ (*weakening*);

  **(A2-I)** $\Vdash^{CPL}_{\Delta} (p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$ (*distribution*);

  **(A3-N)** $\Vdash^{CPL}_{\Delta} (\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p)$;

  **(R1-MP)** $\{p, p \rightarrow q\} \Vdash^{CPL}_{\Delta} q$ (*modus ponens* or *detachment*). □

$CPL$ may also be seen as a non-conservative extension of *minimal intuitionistic logic*, which is generated by schemas **A1-I**, **A2-I** and rule **R1-MP** only.

Our (partial) definition of $CPL$ is slightly unusual. van Dalen (1994) uses a unary logical connective $\bot$ denoting falsehood, which is admittedly not essential. Another important distinction is in relation to the propositional proof calculus adopted by Hilbert and Ackermann (1928), where an additional *uniform substitution* rule is proposed. They mention in that work:

[6]It is worthwhile recalling that Hilbert-style calculi do not have inference rules whereby assumptions can be discharged. This means that $\delta$ as in Definition 2.2.6 is empty for **R1-MP**.

> We may substitute for a sentential variable any sentential combination provided that the substitution is made whenever that sentential variable occurs. (Hilbert and Ackermann 1928)

Since we use *schematic variables* like $\{p, q, r\} \subseteq \mathcal{G}^{CPL}(\Delta)$ in our axiomatisation, substitution would be superfluous here. Our choice also makes *replacement by equivalents* a derivable rule, meaning that it is possible, based on the axiom schemas and inference rules of our proof calculus, to show that an additional rule replacing formulas by logically equivalent ones does not allow us to derive more properties than the original axiomatisation. The statement of this rule, which is often useful in constructing derivations, appears in Appendix I.

In order to ensure that the entailment system above is really well-defined, it is necessary to show that it complies with the generic definition provided in the previous section. We have to prove that propositional signatures and the respective morphisms indeed determine a category. We develop below the proof of this straightforward result just as a matter of completeness. In fact, we show in addition that the category of finite sets has the desirable property of being *finitely co-complete*, which has been identified by Goguen and Burstall (1992) as a necessary condition to support specification in the large:

**Theorem 2.3.2 (Category of Finite Sets)** The collections of finite sets and set-valuated functions define a category **FinSet**. In addition, **FinSet** has both *initial element* and *pushouts*, being in this way *finitely co-complete.*

Proof: Given **FinSet**-morphisms $f$ and $g$, $X \xrightarrow{f} Y$ and $Y \xrightarrow{g} Z$, the function $(g \circ f)(x) \stackrel{\text{def}}{=} g(f(x))$, $x \in X$, is the composition of $f$ and $g$. Considering also a **FinSet**-morphism $h$, $Z \xrightarrow{h} W$, the following diagram commutes (so **ASS** holds):

$$\begin{tikzcd}
& X \arrow[ddl, "f"'] \arrow[ddr, "g\circ f"] \arrow[d, shift right, "(h\circ g)\circ f"'] \arrow[d, shift left, "h\circ(g\circ f)"] & \\
& W & \\
Y \arrow[ur, "h\circ g"'] \arrow[rr, "g"'] & & Z \arrow[ul, "h"']
\end{tikzcd}$$

Moreover, for each $X$ in obj **FinSet**, there is an $X \xrightarrow{\mathbf{id}_X} X$ such that $\forall x \in X \cdot \mathbf{id}_X(x) = x$, the identity function over $X$. Given **FinSet**-morphisms $f$ and $g$, $Y \xrightarrow{f} X$ and $X \xrightarrow{g} Z$, the following diagram commutes (so **ID** holds):

$$\begin{tikzcd}
& X \arrow[d, shift left, "\mathbf{id}_X"] \arrow[dr, "g \circ \mathbf{id}_X"] & \\
Y \arrow[ur, "\mathbf{id}_X \circ f"] \arrow[r, "f"'] & X \arrow[u, shift left, "\mathbf{id}_X"] \arrow[r, "g"'] & Z
\end{tikzcd}$$

The associative and identity axioms obtain, showing that **FinSet** is a category.

We know that $\{\,\}$ belongs to obj **FinSet**. For every $X$ in obj **FinSet**, there is a **FinSet**-morphism $f$, $\{\,\} \xrightarrow{f} X$, such that $\forall y \in \{\,\} \cdot \exists!\, x \in X \cdot f(y) = x$, because this formula characterising *empty functions* holds vacuously. Suppose that there is another $g$ in morph **FinSet**, $\{\,\} \xrightarrow{g} X$ and $\forall y \in \{\,\} \cdot \exists!\, x \in X \cdot g(y) = x$. The extensional definition of function equality says that for each pair $A \xrightarrow{f,g} B$, $f = g$ if and only if $\forall x \in A \cdot f(x) = g(x)$, but for $A = \{\,\}$ this property holds vacuously. This ensures that $f = g$, which means that there is exactly one arrow from $\{\,\}$ to any other set. Thus, $\{\,\}$ is the initial object of **FinSet**.

Assume given the **FinSet**-objects $X$, $Y$, $Z$ and the **FinSet**-morphisms $f_0$, $g_0$, $X \xrightarrow{f_0} Y$ and $X \xrightarrow{g_0} Z$. Construct $W$ in obj **FinSet**, $f_1$ and $g_1$ in morph **FinSet**, $Y \xrightarrow{f_1} W$ and $Z \xrightarrow{g_1} W$, so that:

$$\forall x \in X \cdot (f_1 \circ f_0)(x) = (g_1 \circ g_0)(x) \qquad (2.3.1)$$

$$\forall w \in W \cdot \exists y \in Y \cdot (f_1(y) = w) \vee \exists z \in Z \cdot (g_1(z) = w) \qquad (2.3.2)$$

The set $W = Y \oplus_X Z$ is called the *amalgamated sum* of $Y$ and $Z$ (after possible renaming). For each triple $P = (W', f_2, g_2)$, $W'$ in obj **FinSet** and **FinSet**-morphisms $f_2$, $g_2$ with $Y \xrightarrow{f_2} W'$ and $Z \xrightarrow{g_2} W'$, such that condition (2.3.1) obtains when $f_1$ and $g_1$ are substituted by $f_2$ and $g_2$, there is a $h$ in morph **FinSet**, $W \xrightarrow{h} W'$, such that $f_2 = h \circ f_1$ and $g_2 = h \circ g_1$. This function is defined as:

$$h(x) \overset{\text{def}}{=} \begin{cases} f_2(y) \text{ if } f_1(y) = x \\ g_2(y) \text{ if } g_1(y) = x \end{cases}$$

Indeed, $h$ is a well-defined function, due to condition (2.3.2), which guarantees that every element of the domain $W$ of $h$ has an image in $W'$, and to the fact that $f_2(y) = h(x) = g_2(y)$ whenever $f_1(y) = x = g_1(y)$, which ensures that the image of $h$ is uniquely determined.

If there is another $h'$ in morph **FinSet**, $W \xrightarrow{h'} W'$, obeying the same conditions, then $h' \circ f_1 = h \circ f_1$ and $h' \circ g_1 = h \circ g_1$. In other terms, $\forall y \in Y \cdot (h' \circ f_1)(y) = (h \circ f_1)(y)$ and $\forall z \in Z \cdot (h' \circ g_1)(z) = (h \circ g_1)(z)$. Because of the definition of $W$, $\forall w \in W \cdot h'(w) = h(w)$. Due to extensionality, $h' = h$. So, $h$ is the unique function up to isomorphism making the following diagram commute (the inner diamond is called a *pushout diagram*):

$$\begin{tikzcd}
& Y \arrow[dr, "f_1"] \arrow[drrr, "f_2"] & & & \\
X \arrow[ur, "f_0"] \arrow[dr, "g_0"'] & & W \arrow[rr, dashed, "h"] & & W' \\
& Z \arrow[ur, "g_1"'] \arrow[urrr, "g_2"'] & & &
\end{tikzcd}$$

Using the fact that a category with initial element (e.g. $\{\,\}$) and pushouts (e.g. $Y \xrightarrow{f_1} W \xleftarrow{g_1} Z$ for $Y \xleftarrow{f_0} X \xrightarrow{g_0} Z$) has finite colimits for all finite diagrams (Barr and Wells 1990), we conclude that **FinSet** is finitely co-complete.

■ **(FinSet Category)**

The importance of the proof above is more pragmatic than theoretical. Because the remainder of the thesis is only concerned with finite sets possibly having some additional structure to serve as signatures and with structure-preserving signature morphisms, we can reuse this result to show that each particular category of signatures is finitely co-complete. In practice, this means that herein it will always be possible to take two signatures and compute their composition by identifying the extra-logical symbols they share.

To ensure that the definition of $CPL$ yields a full entailment system, it remains to be shown that the designated properties of each entailment relation are supported by the chosen proof calculus. We show that for any $\{p, q, r\} \subseteq \mathcal{G}^{CPL}(\Delta)$ there is a proof which complies with our axiomatisation and enables us to obtain such properties for each $\Delta$ in $\mathsf{obj}$ **Sig**. This is verified as follows, where (1), (2) and (3) refer to the cases in the definition of $Pr$[7]:

**reflexivity:** From (1) we can infer that $(\{\,\}, p) \in Pr_\Delta(\{p\}, p)$. Moreover, the generic proof of **REFL** : $p \rightarrow p$ stated in Appendix I allows us to say that

[7]Together with **R1-MP**, the following axiomatisation of a *linear implicative calculus* extracted from (Gabbay and de Queiroz 1992) exemplifies Hilbert-style presentations which do not generate full entailment systems:

**(REFL)** $p \rightarrow p$ (reflexivity);

**(PERM)** $(p \rightarrow (q \rightarrow r)) \rightarrow (q \rightarrow (p \rightarrow r))$ (permutation);

**(LTRAN)** $(p \rightarrow q) \rightarrow ((r \rightarrow p) \rightarrow (r \rightarrow q))$ (left-transitivity);

**(RTRAN)** $(p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))$ (right-transitivity).

Namely, if we read $\rightarrow$ as $\multimap$, it is easy to recognise a fragment of *linear logic* (Girard 1987), which lacks monotonicity. By analysing this axiomatisation, we conclude that the usual definition of derivations in Hilbert-style calculi, possibly disconnected linearly ordered sequences of steps, is too strong: every entailment relation so defined is automatically made reflexive and monotonic. That is why connectedness is required in (3) and the entailments generated by derivations containing a single assertion are disregarded by our definition. In this way, only the proposed calculus can help us to prove the properties required in full entailment systems.

there exists $(D, p \rightarrow p) \in Pr_{\Delta}(\{\,\}, p \rightarrow p)$, based on (2). Applying **R1-MP** in (3) using the previous sentences as premises, we conclude that there is a non-empty $D'$ such that $(D', p) \in Pr_{\Delta}(\{p\}, p)$ and, by faithfulness, that $\vdash^{CPL}$ is reflexive;

**monotonicity:** The definition of monotonicity allows us to assume that (i) there is $(D, p) \in Pr_{\Delta}(\Psi_1, p)$, due to the faithfulness condition, and (ii) $\Psi_1 \subseteq \Psi_2$. Choosing any $q \in \Psi_2$, (1) renders (iii) $(\{\,\}, q) \in Pr_{\Delta}(\{q\}, q)$. We also have $(\{\,\}, p \rightarrow (q \rightarrow p)) \in Pr_{\Delta}(\{\,\}, p \rightarrow (q \rightarrow p))$, due to the use of **A1-I** in (2). Using this fact to support two consecutive applications of **R1-MP** in (3), first together with (i) and later with (iii), we can infer that $\Psi_1 \cup \{q\} \vdash_{\Delta} p$ due to faithfulness. After iterating this process for all the other elements of $\Psi_2$ not in $\Psi_1$, taking the outcome of each previous step as the input, we conclude that $\vdash^{CPL}$ is monotonic because of (ii);

**transitivity:** The definition of transitivity allows us to assume that (i) there is $(D_p^1, p) \in Pr_{\Delta}(\Psi_1, p)$ for each $p \in \Psi_2$, and (ii) there is $d = (D_q^2, q) \in Pr_{\Delta}(\Psi_1 \cup \Psi_2, q)$, both due to faithfulness. We show that there is a $d' = (D', q)$, $d \in Pr_{\Delta}(\Psi_1, q)$, obtained from $d$ by recursion. If $d = (\{\,\}, q)$ such that $q \in \Psi_2$, use **A1-I** as in the case of reflexivity to show that $d' = (D_q^1, q) \in Pr_{\Delta}(\Psi_1, q)$ based on (i). If $d = (\{\,\}, q)$ such that $q \in Ax(\Delta) \cup \Psi_1$, $d' = d$. If $d = (\{(D_r, r), (D_{r\rightarrow q}, r \rightarrow q)\}, q)$, apply the same process to $D_r$, $D_{r\rightarrow q}$ and obtain the following $d' = (\{(D'_r, r), (D'_{r\rightarrow q}, r \rightarrow q)\}, q)$:

| | |
|---|---|
| 1. $r$ | $D_r$ |
| 2. $r \rightarrow q$ | $D_{r\rightarrow q}$ |
| 3. $(r \rightarrow q) \rightarrow ((r \rightarrow r) \rightarrow (r \rightarrow q))$ | **LTRAN** |
| 4. $(r \rightarrow r) \rightarrow (r \rightarrow q)$ | **R1-MP** 2, 3 |
| 5. $r \rightarrow r$ | **REFL** |
| 6. $r \rightarrow q$ | **R1-MP** 5, 4 |
| 7. $q$ | **R1-MP** 1, 6 |

where **REFL** and **LTRAN** are verified based on the axiomatisation of $CPL$. Note that this process is applicable to extensions of $CPL$ wherein none of the above is the case. For any other $d = (D_q^1, q)$, apply the same process to each $d' \in D_q^1$ and construct $(D_q^{1'}, q)$ accordingly. Because the first case in the definition of $Pr_{\Delta}$ is the only way of introducing hypotheses in a derivation and we have eliminated all the sentences of $\Psi_2$ from $d$ in $d'$, we conclude that $d' \in Pr_{\Delta}(\Psi_1, q)$. By faithfulness, $\vdash^{CPL}$ is transitive;

**strong-structurality:** Assume that $\Psi \vdash_{\Delta} p$. For each $\Delta \xrightarrow{\tau} \Delta'$ in morph **Sig**, we prove that, if $d = (D, p) \in Pr_{\Delta}(\Psi, p)$ then $d' = (\tau(D), \tau^{\#}(p))|, d \in Pr_{\Delta'}(\mathcal{G}(\tau)(\Psi), \tau^{\#}(p))$, for $\tau^{\#}$ as in Definition 2.2.2. Hence, applying the faithfulness condition twice, we obtain $\mathcal{G}(\tau)(\Psi) \vdash_{\Delta'} \tau^{\#}(p)$, which means that $\vdash^{CPL}$ is strongly structural. Assume $d$ given. Due to the minimality of $Pr_{\Delta}(\Psi, p)$, $d$ corresponds to one of the following cases:

- $d = (\{\,\}, p)$ and $p \in \Psi$. In this case, $d' = (\{\,\}, \tau^{\#}(p)), d \in Pr_{\Delta'}(\mathcal{G}(\tau)(\Psi), \tau^{\#}(p))$ (i.e., assertions of assumptions are translated into similar assertions);
- $d = (\{\,\}, p)$ and $p \in Ax(\Delta)$. The axiomatisation of $CPL$ allows us to say, providing $\{s, t, u\} \subseteq \mathcal{G}^{CPL}(\Delta)$, that $\tau^{\#}(p)$ is either:
  1. $\tau^{\#}(s) \to (\tau^{\#}(t) \to \tau^{\#}(s))$ if $p \in Ax(\Delta)$ because of **A1-I**;
  2. $(\tau^{\#}(s) \to (\tau^{\#}(t) \to \tau^{\#}(u))) \to ((\tau^{\#}(s) \to \tau^{\#}(t)) \to (\tau^{\#}(s) \to \tau^{\#}(u)))$ if $p \in Ax(\Delta)$ because of **A2-I** ;
  3. $(\neg\tau^{\#}(s) \to \neg\tau^{\#}(t)) \to (\tau^{\#}(t) \to \tau^{\#}(s))$ if $p \in Ax(\Delta)$ because of **A3-N**;

  In any case, $\tau^{\#}(p) \in Ax(\Delta')$. So, $d' = (\{\,\}, \tau^{\#}(p)), d \in Pr_{\Delta'}(\mathcal{G}(\tau)(\Psi), \tau^{\#}(p))$ (i.e., instances of schemas are translated into axioms);
- $d = (D, p)$ such that $D \neq \{\,\}$. Apply the same process above to each $d_i \in D$ and obtain $\tau(D)$. Due to the axiomatisation of $CPL$, $\tau(D)$ must have the form $\{(D_{\tau^{\#}(q)}, \tau^{\#}(q)), (D_{\tau^{\#}(q)\to\tau^{\#}(p)}, \tau^{\#}(q) \to \tau^{\#}(p))\}$. By applying **R1-MP** in (3) we obtain $d' \stackrel{\text{def}}{=} (\tau(D), \tau^{\#}(p)) \in Pr_{\Delta'}(\mathcal{G}(\tau)(\Psi), \tau^{\#}(p))$ (i.e., applications of inference rules are translated accordingly).

The verification of the result above was not developed in the most economical way. It was devised as such not only to spot a set of provable theorems which ensure the properties of full entailment systems, but also to shed some light on what requires attention in developing similar results for other systems. Reflexivity, monotonicity and transitivity do not demand all the axiom schemas of $CPL$ and are still valid considering some weaker axiomatisations. On the other hand, if the schemas and rules of $CPL$ are stated based on the grammar of an extended Hilbert-style calculus, the properties above do not need to be examined again because their verification does not depend on the additional structure of the proof calculus. We adopt this rationale to argue that all the entailment systems described in the sequel are reflexive, monotonic and transitive.

The case of structurality is more complex. To prove that $CPL$ is strongly structural, we had to examine all the applications of inference rules and instances of axiom schemas showing that they are translated into similar constructions based on the target signature of each morphism. This guarantees that all the theorems of each $CPL$ presentation are translated by each morphism into theorems of its target presentation. Even after having verified this result for the axiomatisation of $CPL$, we are still obliged to prove the same for the remaining schemas and rules of each proof calculus wherein this axiomatisation appears embedded. To simplify this task, it is enough to show that no schema or rule strictly depends on the symbols existing in the underlying signature. To understand why, note that the only possible distinction between the theories of a presentation and of its translation along a morphism may appear because some theorems are generated by schemas or rules stated in terms of the set of symbols in the original signature, which can be expanded by a morphism. Thus, the corresponding theorems in the theory of the target presentation would not exist. We shall study a weakly structural entailment system in the next chapter.

Even though classical propositional logic is not highly expressive, the functionality of real systems can already be represented in specifications to some extent. Suppose that $CPL$ is to be applied in the design of a replacement for a mechanical system presently in use by a supermarket. The main purpose of the system is to prevent trolleys from being stolen. To be allowed to use a trolley, each customer is required to leave a special purpose identification card as a deposit in a safe so as to release the attached trolley immediately. As soon as a trolley is locked again to the system, the card can be collected.

We adopt here a design discipline prescribing the representation of each object in a problem domain as a separate theory presentation, following in this way Fiadeiro and Maibaum (1992). In the supermarket system, it is easy to identify these objects as the locker, the safe and the mechanical device which obliges the first two objects to behave in a coordinated manner. In Figure 2.6, each of these objects is described by a specification consisting of a signature and a set of propositional axioms. The propositional symbols in each signature denote both the state and the instantaneous events occurring in the system. Axioms define how these entities are related to each other. For instance, according to (4.1), which constrains the occurrence of an action according to an attribute value, only when a trolley is currently locked to the system can it be released. Another example is provided by (2.1), saying that the occurrence of actions *push* and *pull* is mutually exclusive. This separation of signature symbols into attributes and actions is regarded in this chapter just as syntactic sugar to make specifications more readable, meaning that at this point these families of symbols do not have any distinguished logical role.

**Specification** DEVICE
**actions** *push*, *pull*
**axioms**
*push* → ¬*pull* (2.1)
**End**

**Specification** SAFECPL
**attributes** *card_in*
**actions** *deposit*, *collect*
**axioms**
*collect* → *card_in* (3.1)
*deposit* → ¬*collect* (3.2)
**End**

**Specification** LOCKERCPL
**attributes** *trolley_in*
**actions** *release*, *lock*
**axioms**
*release* → *troley_in* (4.1)
*release* → ¬*lock* (4.2)
**End**

**Specification** SYSTEMCPL
**actions** *use*, *return*
**attributes** *trolley_in*, *card_in*
**axioms**
*use* → ¬*return* (5.1)
*use* → *trolley_in* (5.2)
*return* → *card_in* (5.3)
**End**

Figure 2.6: Specification of the supermarket system in $CPL$.

(a)

$$\begin{tikzcd}
 & \text{SYSTEMCPL} & \\
\text{LOCKERCPL} \arrow[ur, "\sigma_1^{\#}"] & & \text{SAFECPL} \arrow[ul, "\sigma_2^{\#}"'] \\
 & \text{DEVICE} \arrow[ul, "\tau_1^{\#}"] \arrow[ur, "\tau_2^{\#}"'] &
\end{tikzcd}$$

(b) LOCKERCPL ⊕ SAFECPL

$$\begin{tikzcd}
release \arrow[r, mapsto, "\sigma_1"] & use & deposit \arrow[l, mapsto, "\sigma_2"'] \\
lock \arrow[r, mapsto, "\sigma_1"] & return & collect \arrow[l, mapsto, "\sigma_2"'] \\
trolley\_in \arrow[r, mapsto, "\sigma_1"] & trolley\_in & \\
 & card\_in & card\_in \arrow[l, mapsto, "\sigma_2"']
\end{tikzcd}$$

$$\begin{tikzcd}
\text{LOCKERCPL} & \text{DEVICE} & \text{SAFECPL} \\
release & push \arrow[l, mapsto, "\tau_1"'] \arrow[r, mapsto, "\tau_2"] & deposit \\
lock & pull \arrow[l, mapsto, "\tau_1"'] \arrow[r, mapsto, "\tau_2"] & collect
\end{tikzcd}$$

Figure 2.7: Configuration of the supermarket system in $CPL$.

We use specification morphisms to connect distinct modular theory presentations and construct in an incremental way descriptions of complex systems. The signature morphisms in Figure 2.7 (b) describe how the specification symbols above are related to each other by way of translation. It is easy to see there that the same set of real events is represented by the pairs of action symbols in each specification, because they are equalised by the morphisms in the diagram. Indeed, when the trigger of the mechanical device is pushed, the trolley is released and the card deposited inside the safe. In effect, they correspond to the same complex event. When the translations above are applied in a compositional manner to axioms, specification morphisms are induced defining a way of putting the set of specifications together to represent the whole system, as presented in part (a). Axiom (2.1) is translated not only into (3.2) but also into (4.2), justifying the fact that the actions of both LockerCPL and SafeCPL are mutually exclusive.

Identifying the same symbol with those of other specifications, we ensure that this symbol will represent a shared resource when the specifications are collapsed into a single object. In our example, all the action symbols of Device, *pull* and *push*, are associated to the symbols of LockerCPL and SafeCPL, because the entire device is a shared object. When we require in addition that constructions like LockerCPL $\xrightarrow{\sigma_1^{\#}}$ SystemCPL $\xleftarrow{\sigma_2^{\#}}$ SafeCPL be *co-limits*, a generalised form of pushout possibly connecting several objects in a co-cone diagram, we do not need to be concerned with the exact definition of the resulting entities because it is provided up to isomorphism. In our example, SystemCPL is only a representative of the class of theory presentations induced by the connection of LockerCPL and SafeCPL through Device and the given morphisms. Any other object in this class could be used in its place and the same is true concerning the morphisms $\sigma_1^{\#}$ and $\sigma_2^{\#}$. Of course, we are only allowed to use such $CPL$ constructions, and we always do so hereafter, because we know they exist, due to the co-completeness of **FinSet**.

So far, we have concentrated on showing that $CPL$ is a useful specification tool. In the formal design of software systems, we also face the problem of verifying characteristic properties. For instance, we may want to prove that the system described above will allow some customer action only if either a trolley or a card is currently held by the system. This can be stated as follows:

$$\vdash_{\text{SystemCPL}} use \lor return \rightarrow trolley\_in \lor card\_in \tag{2.3.3}$$

We use the structure of SystemCPL and the specification axioms to verify this property. We also rely on helpful theorems and rules stated in Appendix I:

Proof:

| | | |
|---|---|---|
| 1. | $release \rightarrow trolley\_in$ | 4.1 (LockerCPL) |
| 2. | $use \rightarrow trolley\_in \vee card\_in$ | **OR-R** $\sigma_1^{\#}(1)$ (SystemCPL) |
| 3. | $collect \rightarrow card\_in$ | 3.1 (SafeCPL) |
| 4. | $return \rightarrow trolley\_in \vee card\_in$ | **OR-R** $\sigma_2^{\#}(3)$ (SystemCPL) |
| 5. | $use \vee return \rightarrow trolley\_in \vee card\_in$ | **OR-L** 2, 4 (SystemCPL) ■ |

The application above of induced specification morphisms to translate the consequence obtained in each proof step is worth noticing. It is in this way that we can enlarge the language of LockerCPL and apply the rule **OR-R** to include an additional disjunct in the right hand side of the first implicative assertion. The resulting sentence belongs to the theory of SystemCPL. Much in the same way, a similar conclusion is obtained from the SafeCPL axiom. If we consider that this verification process started from (2.3.3), we are also allowed to say that the proof of that property was decomposed into a set of proofs of simpler properties by the proof calculus and the given morphisms. This way of decomposing proofs was first studied by Fiadeiro and Maibaum (1992).

It is interesting to note that the morphisms employed above in the configuration of the system are all faithful. For instance, because (2.1) belongs to Device, all the properties involving the symbols in this presentation when translated by $\tau_1$ into LockerCPL can already be derived within Device. This means that the object does not have more properties when placed in the complex configuration. It is useful to leave the possibility of using non-faithful morphisms open so that some properties of specified objects emerge only when they are placed in certain configurations. There are two ways of supporting this feature: to use looser specifications (with a weaker set of axioms) or to adopt a weakly structural entailment system. Both cases shall be exploited in the remainder of the thesis.

## 2.4 Propositional Linear Time Logic

We discovered in the previous section that classical propositional logic is useful in dealing with the finite state and the relations between instantaneous events of real systems. However, the same logic turns out to be less useful to represent change and time dependent behaviour. Essentially, it would be necessary to code the passing of time in each *CPL* theory so that we could rely on this feature. Unfortunately, *CPL* as defined above is not expressive enough to permit the representation of infinite flows of time. To overcome this and other limitations, temporal logics having additional connectives to deal with the time dimension may be used. Here we choose an entailment system with two temporal connectives only: **beg**, denoting the beginning of time, and $\mathbf{V}$, the strict strong until connective which is used to express, when we assert $p\mathbf{V}q$, that the property $p$ occurs strictly in the future (i.e., after the current moment) and $q$ happens uninterruptedly from the next instant until but not necessarily including the moment of the $p$ occurrence[8].

**Definition 2.4.1 (Propositional Linear Time Logic)** The entailment system of *propositional linear time logic*, $PLTL$ for short, is defined as follows:

- $\mathbf{Sig}^{PLTL} \cong \mathbf{Sig}^{CPL}$;
- $\mathcal{L}^{PLTL} \overset{\text{def}}{=} \mathcal{L}^{CPL} \cup \{\mathbf{beg}, \mathbf{V}\}$[9];
- $\mathcal{E}^{PLTL} \cong \mathbf{id}_{\mathbf{Sig}^{PLTL}}$;
- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{PLTL}$, $\mathcal{G}^{PLTL}(\Delta)$ is defined by $P^{PLTL}$ as follows:

  $$P^{PLTL} ::= P^{CPL} \mid \mathbf{beg} \mid (P^{PLTL})\mathbf{V}(P^{PLTL})$$

  We use the following abbreviations for each $\{p,q\} \subseteq \mathcal{G}^{PLTL}(\Delta)$ to introduce the connectives next, the non-strict strong until, eventually in the future, always in the future and weak until (apart from next, these connectives all range over the present moment as well):

  **(D6-X)** $\mathbf{X}p \overset{\text{def}}{=} p\mathbf{V}\bot$;

  **(D7-U)** $p\mathbf{U}q \overset{\text{def}}{=} q \vee (p \wedge q\mathbf{V}p)$;

  **(D8-F)** $\mathbf{F}p \overset{\text{def}}{=} \top\mathbf{U}p$ [or $p \vee p\mathbf{V}\top$];

  **(D9-G)** $\mathbf{G}p \overset{\text{def}}{=} \neg\mathbf{F}(\neg p)$ [or $p \wedge \neg(\neg p)\mathbf{V}\top$];

  **(D10-W)** $p\mathbf{W}q \overset{\text{def}}{=} \mathbf{G}p \vee p\mathbf{U}q$ [or $(p \wedge \neg(\neg p)\mathbf{V}\top) \vee q \vee (p \wedge q\mathbf{V}p)$];

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{PLTL}$, the entailment relation $\vdash_{\Delta}^{PLTL}$ is generated by the proof calculus of $CPL$ together with the following one, provided that they are both stated over $\mathcal{G}^{PLTL}(\Delta)$, wherein $p$, $q$, $r$ and $s$ are included:

  **(A4-GV)** $\Vdash_{\Delta}^{PLTL} \mathbf{G}(p \to q) \to (p\mathbf{V}r \to q\mathbf{V}r)$;

  **(A5-GV)** $\Vdash_{\Delta}^{PLTL} \mathbf{G}(p \to q) \to (r\mathbf{V}p \to r\mathbf{V}q)$;

[8] $p\mathbf{V}q \ \equiv\ \mathbf{V}(p,q)$ (Gabbay *et al.* 1994) $\equiv\ q\hat{\mathbf{U}}p$ (Manna and Pnueli 1989).

[9] We have chosen $\mathbf{V}$ to distinguish the strict strong until connective proposed by Kamp from $\mathbf{U}$, the non-strict connective normally found in temporal logics of programs (Pnueli 1977).

**(A6-V)** $\Vdash^{PLTL}_{\Delta} p\mathbf{V}q \rightarrow p\mathbf{V}(q \wedge p\mathbf{V}q)$;

**(A7-V)** $\Vdash^{PLTL}_{\Delta} (p \wedge q\mathbf{V}p)\mathbf{V}p \rightarrow q\mathbf{V}p$;

**(A8-V)** $\Vdash^{PLTL}_{\Delta} p\mathbf{V}q \wedge r\mathbf{V}s \rightarrow (p\wedge r)\mathbf{V}(q\wedge s)\vee(p\wedge s)\mathbf{V}(q\wedge s)\vee(q\wedge r)\mathbf{V}(q\wedge s)$;

**(A9-V)** $\Vdash^{PLTL}_{\Delta} (p \vee q)\mathbf{V}r \rightarrow p\mathbf{V}r \vee q\mathbf{V}r$;

**(A10-G)** $\Vdash^{PLTL}_{\Delta} \mathbf{G}(p \rightarrow \mathbf{X}p) \rightarrow (p \rightarrow \mathbf{G}p)$;

**(A11-X)** $\Vdash^{PLTL}_{\Delta} \mathbf{X}\top$;

**(A12-Xbeg)** $\Vdash^{PLTL}_{\Delta} \neg\mathbf{X}(\mathbf{beg})$;

**(R2-G)** $\{p\} \Vdash^{PLTL}_{\Delta} \mathbf{G}p$;

**(R3-begG)** $\{\mathbf{beg} \rightarrow \mathbf{G}p\} \Vdash^{PLTL}_{\Delta} p$. □

Note that, by including above the set of axioms of classical propositional logic, we obtain a proof calculus substantially different from that proposed by Manna and Pnueli (1989), where all the propositional validities are accepted without presentation of formal proof. Our axiomatic presentation appears to be more appropriate given our additional interest in formal stepwise development, where only formal reasoning can justify software constructions in full.

The proof calculus above is obtained from sets of axioms which also consider a strong strict since connective as discussed in (Gabbay *et al.* 1994) by removing this past-time connective and including **beg** instead. Schemas **A4**, **A5** and **A9** together with **R2** guarantee that we have a normal modal logic, which can be interpreted over relational structures. **A6-7** ensure the transitivity of these relations and we enter in this way the realm of temporal logic. **A8** in the presence of the other axioms implies that time is linearly ordered towards the future. In particular, due to our choice of initialised time flows, this is true everywhere. We also include **A10** to capture temporal induction. We use **A11** not only to guarantee that the time flow does not have endpoints but also to ensure that there is always a next instant, capturing discrete time. Axiom **A12** says that no instant precedes the initial one. Rule **R2** is the usual temporal generalisation and **R3** may be called **begG**-elimination.

The reader may want to verify that **A1-11** and **R1-2** entail all the propositional theorems of the logical consequence relation defined by Manna and Pnueli (1983), which is stated in terms of the set of connectives defined as abbreviations here. This lengthy proof can be developed based on the auxiliary theorems in Appendix I. We adopt flows of time with fixed characteristics as in their work to minimise the possibility of generating inconsistent composed specifications. This would be the case if two composed specifications could assume respectively discrete and dense flows, with and without endpoints, and so on. It is easy to see that, adopting the sufficiently general class of initialised discrete flows without end points, we can still talk about most interesting properties in terms of the occurrence of actions. Termination, say, can be satisfactorily represented by the eventual and everlasting impossibility of action occurrence.

The application of linear time logical systems in software design has been streamlined by the separation of temporal properties into two distinct families due to Alpern and Schneider (1985) and the respective development of suitable reasoning principles by a number of authors. *Liveness* properties stating what a system eventually performs offer great challenges to verification methods. They are treated using the general proof rule derived in Section 2.7. *Safety* properties, which define what a system always ensures, are verified here using the following derived inference rule:

**Theorem 2.4.2 (Inference Rule IND-begG)** The following inference rule for any $p \in \mathcal{G}^{PLTL}(\Delta)$ is derivable in $PLTL$:

**(IND-begG)** $\{\mathbf{beg} \rightarrow p, \mathbf{G}(p \rightarrow \mathbf{X}p)\} \Vdash_{\Delta}^{PLTL} p$ (*anchored temporal induction*).

Proof:

| | | |
|---|---|---|
| 1. | $\mathbf{beg} \rightarrow p$ | **Ass** |
| 2. | $\mathbf{G}(p \rightarrow \mathbf{X}p)$ | **Ass** |
| 3. | $\mathbf{G}(p \rightarrow \mathbf{X}p) \rightarrow (p \rightarrow \mathbf{G}p)$ | **A10-G** |
| 4. | $p \rightarrow \mathbf{G}p$ | **R1-MP** 2, 3 |
| 5. | $\mathbf{beg} \rightarrow \mathbf{G}p$ | **HS** 1, 4 |
| 6. | $p$ | **R3-begG** 5 ■ **(IND-begG)** |

where **HS** is the *hypothetical syllogism* rule stated in Appendix I. It is important to stress that anchored temporal induction must be captured as a proof rule since this property cannot be consistently written as an axiom schema in the presence of the other usual temporal logic schemas. Kröger (1987) recalls that adopting a similar schema would trivialise the whole logic. Manna and Pnueli (1989) overcome this problem as above, considering that **beg** is definable in terms of past time connectives. In $TLA$, the Temporal Logic of Actions of Lamport (1994), an invariance rule is adopted instead since **beg** has no logical counterpart and each canonical specification defines an initialisation condition.

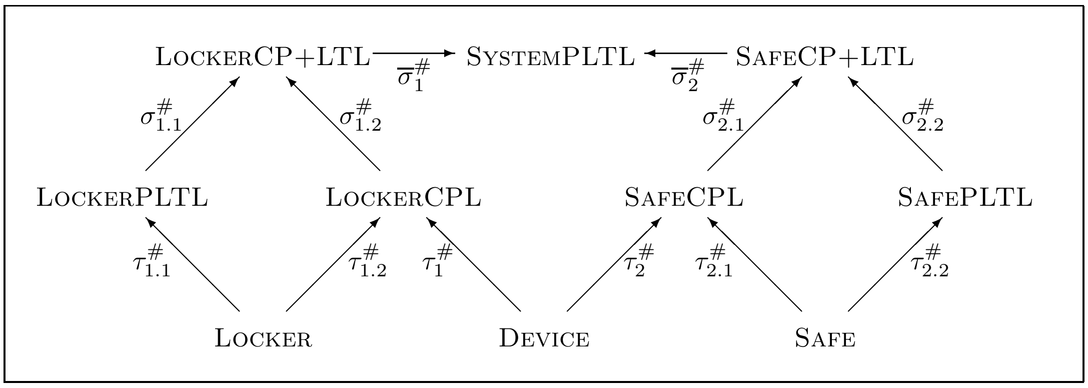
Let us return to our supermarket example. We can now increment the specification of the system using the features of temporal logic. Note that all the previous specifications can be reused because $CPL$ formulas are allowed in $PLTL$. Due to this fact, we present the extended definition of the system in less detail. In order to define that a state $p$ of the system changes, sometimes according to the occurrence of specific actions, we use the following definition:

$$Mod(p) \stackrel{\text{def}}{=} (p \wedge \mathbf{X}(\neg p)) \vee (\neg p \wedge \mathbf{X}p)$$

**Specification** LockerPLTL
**attributes** *trolley_in*
**actions** *release*, *lock*
**axioms**
**beg** $\rightarrow$ *troley_in* (6.1)
*release* $\rightarrow \mathbf{X}(\neg trolley\_in)$ (6.2)
*lock* $\rightarrow \mathbf{X}(trolley\_in)$ (6.3)
*release* $\vee$ *lock* $\vee \neg Mod(trolley\_in)$ (6.4)
**End**

**Specification** SafeLPTL
**attributes** *card_in*
**actions** *deposit*, *collect*
**axioms**
**beg** $\rightarrow \neg card\_in$ (7.1)
*deposit* $\rightarrow \mathbf{X}(card\_in)$ (7.2)
*collect* $\rightarrow \mathbf{X}(\neg card\_in)$ (7.3)
*deposit* $\vee$ *collect* $\vee \neg Mod(card\_in)$ (7.4)
**End**

Figure 2.8: Specification of the supermarket system in $PLTL$.



$$\begin{tikzcd}
& \text{LockerCP+LTL} \arrow[rrr, swap, "\sigma_1^\#"] & & & \text{SystemPLTL} & & & \text{SafeCP+LTL} \arrow[lll, "\sigma_2^\#"] & \\
\text{LockerPLTL} \arrow[ur, "\sigma_{1.1}^\#"] & & \text{LockerCPL} \arrow[ul, swap, "\sigma_{1.2}^\#"] & & & & \text{SafeCPL} \arrow[ur, "\sigma_{2.1}^\#"] & & \text{SafePLTL} \arrow[ul, swap, "\sigma_{2.2}^\#"] \\
& \text{Locker} \arrow[ul, "\tau_{1.1}^\#"] \arrow[ur, swap, "\tau_{1.2}^\#"] & & & \text{Device} \arrow[ull, "\tau_1^\#"] \arrow[urr, swap, "\tau_2^\#"] & & & \text{Safe} \arrow[ul, "\tau_{2.1}^\#"] \arrow[ur, swap, "\tau_{2.2}^\#"] &
\end{tikzcd}$$

Figure 2.9: Configuration of the supermarket system in $PLTL$.

This abbreviation is employed in the axioms of Figure 2.8. We also assume the existence of specifications Safe and Locker containing only the signature symbols presented in Section 2.3. These are used to define the extended configuration of the system, which appears in Figure 2.9. We only mention in that figure the relevant specifications because the other ones are defined up to isomorphism by the pushout construction which results in SystemPLTL. In addition, we postulate that the morphisms remaining to be defined are all identities.

The connectives of temporal logic allow us to make reference to the passing of time in each specification. Using **beg** in axiom (6.1), we define that a trolley is initially attached to the locker. Conversely, the safe is originally empty according to (7.1). The effect of actions over the attributes of each entity are defined based on **X**, the next time connective. Axiom (6.2) specifies that a trolley will not be kept locked to the system in the next instant if it is released in the current moment. Once locked again to the system, (6.3) ensures that the trolley will be subsequently available. Although this kind of axiom treats action effects with precision, they do not ensure that the respective attributes will remain invariant otherwise. This is normally called the *frame problem* in the literature, which becomes overly complicated in the presence of concurrency. Because in our example each object does not present internal concurrency, their actions being mutually exclusive, we may adopt a simple solution. Ryan *et al.* (1991) propose an axiom requiring that either the actions of each object happen or else the attribute values do not change. This is what (7.4) says: that a card is either deposited or collected at each moment or else the state of the safe is not modified. In Chapter 3, we will capture this notion of locality logically.

We have verified that the supermarket system will not allow any customer action unless some object is held by the system, be it a trolley or a card. It is also possible to prove using *CPL* that this property can be made stronger in that precisely one object must be held if any action is to take place. Using our temporal proof calculus and the extended specification of the system, we can now prove that this state condition is never violated. That is, it is always the case that either a trolley or a card is connected to the system:

$$\vdash_{\textsc{SystemPLTL}} \mathbf{G}(trolley\_in \leftrightarrow \neg card\_in) \tag{2.4.1}$$

Simple temporal reasoning based on the theorems in Appendix I shows that this property can be decomposed within SystemPLTL:

1. $\mathbf{G}(trolley\_in \rightarrow \neg card\_in) \wedge \mathbf{G}(card\_in \rightarrow \neg trolley\_in)$ **Ass**
2. $(\mathbf{G}(trolley\_in \rightarrow \neg card\_in) \wedge \mathbf{G}(card\_in \rightarrow \neg trolley\_in)) \leftrightarrow$ **DIST-ANDG**
   $\mathbf{G}((trolley\_in \rightarrow \neg card\_in) \wedge (card\_in \rightarrow \neg trolley\_in))$
3. $(\mathbf{G}(trolley\_in \rightarrow \neg card\_in) \wedge \mathbf{G}(card\_in \rightarrow \neg trolley\_in)) \rightarrow$ **IFF-E** 2
   $\mathbf{G}((trolley\_in \rightarrow \neg card\_in) \wedge (card\_in \rightarrow \neg trolley\_in))$
4. $\mathbf{G}((trolley\_in \rightarrow \neg card\_in) \wedge (card\_in \rightarrow \neg trolley\_in))$ **R1-MP** 1, 3
5. $\mathbf{G}(trolley\_in \leftrightarrow \neg card\_in)$ **D5-IFF** 4

Because the two conjuncts in (1) are similar, we will only develop the proof of one of these properties. The other proof can be developed similarly.

At this point we have a good opportunity to apply our derived inference rule **IND-begG** for anchored temporal induction. This inference rule allows us to decompose the proof of the assumption above into the verification of an initial condition and an invariance formula of SystemPLTL:

6. $\mathbf{beg} \rightarrow (trolley\_in \rightarrow \neg card\_in)$ **Ass**
7. $(trolley\_in \rightarrow \neg card\_in) \rightarrow \mathbf{X}(trolley\_in \rightarrow \neg card\_in)$ **Ass**
8. $\mathbf{G}((trolley\_in \rightarrow \neg card\_in) \rightarrow \mathbf{X}(trolley\_in \rightarrow \neg card\_in))$ **R2-G** 7
9. $trolley\_in \rightarrow \neg card\_in$ **IND-begG** 6, 8
10. $\mathbf{G}(trolley\_in \rightarrow \neg card\_in)$ **R2-G** 9

The initial condition (6) is proved using classical reasoning:

| | | |
|---|---|---|
| 11. | $\mathbf{beg} \rightarrow \mathit{trolley\_in}$ | (6.1) |
| 12. | $\mathbf{beg} \rightarrow \neg \mathit{card\_in}$ | (7.1) |
| 13. | $\mathbf{beg} \rightarrow \mathit{trolley\_in} \wedge \neg \mathit{card\_in}$ | **AND-R** $\rho_1(11)$, $\rho_2(12)$ |
| 14. | $\neg \mathit{card\_in} \rightarrow (\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **A1-I** |
| 15. | $(\mathit{trolley\_in} \wedge \neg \mathit{card\_in}) \rightarrow (\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **AND-L** 14 |
| 16. | $\mathbf{beg} \rightarrow (\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **HS** 13, 15 |

where $\rho_i \overset{\text{def}}{=} (\overline{\sigma}_i \circ \sigma_{i.i})^{\#}$, the composition of two morphisms determined up to isomorphism by the pushout construction in Figure 2.9. The proof of the invariance formula (7) requires additional temporal reasoning:

| | | |
|---|---|---|
| 17. | $\mathit{use} \rightarrow \mathbf{X}(\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **Ass** |
| 18. | $\mathit{return} \rightarrow \mathbf{X}(\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **Ass** |
| 19. | $\neg Mod(\mathit{trolley\_in}) \wedge \neg Mod(\mathit{card\_in}) \rightarrow \mathbf{X}(\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **Ass** |
| 20. | $\mathit{use} \vee \mathit{return} \vee (\neg Mod(\mathit{trolley\_in}) \wedge \neg Mod(\mathit{card\_in})) \rightarrow$ $\mathbf{X}(\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **OR-L** 17, 18, 19 |
| 21. | $\mathit{release} \vee \mathit{lock} \vee \neg Mod(\mathit{trolley\_in})$ | (6.4) |
| 22. | $\mathit{deposit} \vee \mathit{collect} \vee \neg Mod(\mathit{card\_in})$ | (7.4) |
| 23. | $(\mathit{use} \vee \mathit{return} \vee \neg Mod(\mathit{trolley\_in})) \wedge$ $(\mathit{use} \vee \mathit{return} \vee \neg Mod(\mathit{card\_in}))$ | **AND-I** $\rho_1(21)$, $\rho_2(22)$ |
| 24. | $\mathit{use} \vee \mathit{return} \vee (\neg Mod(\mathit{trolley\_in}) \wedge \neg Mod(\mathit{card\_in}))$ | **DM**, **IFF-E**, **R1-MP** 23 |
| 25. | $\mathbf{X}(\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **R1-MP** 24, 20 |
| 26. | $\mathbf{X}(\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in}) \rightarrow$ $((\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in}) \rightarrow \mathbf{X}(\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in}))$ | **A1-I** |
| 27. | $(\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in}) \rightarrow \mathbf{X}(\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **R1-MP** 25, 26 |

In order to complete the verification of property (2.4.1), we prove assumption (17) above as follows:

| | | |
|---|---|---|
| 28. | $\mathit{use} \rightarrow \mathbf{X}(\mathit{card\_in})$ | (7.2) |
| 29. | $\mathit{use} \rightarrow \mathbf{X}(\neg \mathit{trolley\_in})$ | (6.1) |
| 30. | $\mathit{use} \rightarrow \mathbf{X}(\mathit{card\_in}) \wedge \mathbf{X}(\neg \mathit{trolley\_in})$ | **AND-R** $\rho_2(28)$, $\rho_1(29)$ |
| 31. | $\mathbf{X}(\mathit{card\_in}) \wedge \mathbf{X}(\neg \mathit{trolley\_in}) \rightarrow \mathbf{X}(\mathit{card\_in} \wedge \neg \mathit{trolley\_in})$ | **DIST-ANDX**, **IFF-E** |
| 32. | $\mathit{use} \rightarrow \mathbf{X}(\mathit{card\_in} \wedge \neg \mathit{trolley\_in})$ | **HS** 30, 31 |
| 33. | $(\mathit{card\_in} \rightarrow \neg \mathit{trolley\_in}) \rightarrow (\neg\neg \mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **CONP** |
| 34. | $\mathit{trolley\_in} \rightarrow \neg\neg \mathit{trolley\_in}$ | **DOUB**, **A3-N**, **R1-MP** |
| 35. | $(\mathit{trolley\_in} \rightarrow \neg\neg \mathit{trolley\_in}) \rightarrow$ $((\neg\neg \mathit{trolley\_in} \rightarrow \neg \mathit{card\_in}) \rightarrow (\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in}))$ | **LTRAN** |
| 36. | $(\neg\neg \mathit{trolley\_in} \rightarrow \neg \mathit{card\_in}) \rightarrow (\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **R1-MP** 34, 35 |
| 37. | $(\mathit{card\_in} \rightarrow \neg \mathit{trolley\_in}) \rightarrow (\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **HS** 33, 36 |
| 38. | $(\mathit{card\_in} \wedge \neg \mathit{trolley\_in}) \rightarrow (\mathit{card\_in} \rightarrow \neg \mathit{trolley\_in})$ | **A1-I**, **AND-L** |
| 39. | $\mathbf{G}((\mathit{card\_in} \wedge \neg \mathit{trolley\_in}) \rightarrow (\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in}))$ | **HS** 38, 37, **R2-G** |
| 40. | $\mathbf{X}(\mathit{card\_in} \wedge \neg \mathit{trolley\_in}) \rightarrow \mathbf{X}(\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **MON-GX**, **R1-MP** 39 |
| 41. | $\mathit{use} \rightarrow \mathbf{X}(\mathit{trolley\_in} \rightarrow \neg \mathit{card\_in})$ | **HS** 32, 40 ■ |

The proof of (18) is developed in the same way, without steps similar to (32-37) because we obtain in this case the required implication in the right direction. Assumption (19) is proved by simple although tedious temporal reasoning, which is omitted here. This concludes the verification of (2.4.1).

An analogy between the properties of concurrent systems and (2.4.1) appears to be in order here. The components of a real system are said to be in a *deadlock* state if and only if it is impossible for each of them to perform computations because the other components have not provided some locally required functionality. Since all the components remain waiting for one another, the whole system stops. A typical example is a circular traffic jam in which no car is allowed to proceed because vehicles in perpendicular streets block the passage. The negation of (2.4.1) is another example wherein either the empty safe will always expect a forbidden action from the empty locker and vice versa, or else both occupied equipment wait forever for their impossible utilisation. Safety properties like deadlock freedom assert that something bad never happens. We have just applied a method which allows us to verify such properties when the given specifications are informative enough.

## 2.5 Propositional Branching Time Logic

Even though propositional linear time logics as studied in the previous section are appropriate for describing the behaviour of some concurrent systems, these logics have limitations too. Sistla *et al.* (1984) and Koymans (1987) proved many inexpressibility results stating that it is impossible to describe unbounded message buffers using such logics. As a practical result, we can infer that the specification and verification of most kinds of concurrent asynchronous message passing systems would require a first-order temporal logic or similar. We shall revisit this issue in the sequel.

There is, however, an alternative direction towards increasing the expressive power of propositional linear time logics that appears to be worthwhile studying at this point. This study is motivated by practical reasons related to message passing systems as well. In this domain, it is frequent to demand forms of guaranteed delivery wherein each dispatched message must be received whenever it becomes possible often enough for the recipient to accept it, a particular kind of liveness property usually called *fairness* (Gabbay *et al.* 1980). Such requirements rule out allegedly unreasonable or unfair behaviours in which some messages are always ignored even though the recipient could accept them. Although we know how to specify that something will happen in the future using the connective **F**, we do not know how to express possibility using a pure linear time logic — that there are some behaviours in which an event indeed occurs despite the fact that we cannot ensure it is the current behaviour. Assuming branching flows of time makes the solution of this problem conceivable.

Two plausible and completely distinct views concerning the definition of branching time logics are available in the literature (Zanardo 1996). The so-called Priorean view advocates that a sentence asserting the eventual occurrence of a proposition is true at a given moment $x$ if and only if the proposition is true at some moment in some future of $x$. Conversely, the Ockhamist view argues that it is meaningless to discuss the truth value of a proposition unless additional information is provided about the actual future. To clarify this distinction, a metaphor can be defined. Assume that a system and two omniscient observers are given, Eager and Lazy. Both see the system evolving almost as defined in the previous section in that each behaviour has an initial instant, is discrete and infinite. Eager, who adopts a Priorean view, politely ignores everything else he knows and follows the system closely, allowing his own current moment of time to be always equal to that of the system. According to his perceptions, what will happen in the future spans as many branches of undetermined possibilities. Lazy, on the contrary, adopts an Ockhamist view and prefers to prevent his time from passing, staying outside of any existing behaviour in the underlying time frame. He can only see the distinct behaviours of the system as a set of linear terminated sequences. For him, what could have otherwise been the case at some moment of a behaviour is defined in terms of other possible behaviours of the system. Comparing these two distinct views, we can conclude that what is regarded as a branching time logic depends on the chosen kind of observer. Both are reasonable views that allow us to talk about possibility.

Axiomatisations of Priorean and Ockhamist logics have different virtues. Priorean logics have been preferred in the study of linguistic structures. Some of these logics, which are normally defined by a reduced set of axiom schemas and rules, are studied in (Gabbay *et al.* 1994). Ockhamist logics have been prevalent in the design of software systems as shown by the extensive literature (Emerson 1990, Stirling 1992, Zanardo and Carmo 1993). This may be due to the fact that all the axiom schemas defining linear time are still valid concerning each behaviour. Priorean logics, on the other hand, do not obey schemas like **A8-V** requiring linearity. Ockhamist branching time demands in this way an additional connective **E** to express possibility, the existence of a potentially distinct behaviour obeying a given property with a strict past history identical to that of the current behaviour. As a result, a larger set of axiom schemas is required. Here we adopt **A**, the dual to **E**, as a logical connective of necessity and choose to define our branching time logic as follows:

**Definition 2.5.1 (Propositional Branching Time Logic)** The entailment system of *propositional branching time logic*, $PBTL$ for short, is defined as follows:

- $\mathbf{Sig}^{PBTL} \cong \mathbf{Sig}^{PLTL}$;
- $\mathcal{L}^{PBTL} \overset{\text{def}}{=} \mathcal{L}^{PLTL} \cup \{\mathbf{A}\}$;
- $\mathcal{E}^{PBTL} \cong \mathbf{id}_{\mathbf{Sig}^{PBTL}}$;
- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{PPBTL}$, $\mathcal{G}^{PBTL}(\Delta)$ is defined by $P^{PBTL}$ as follows:

  $$P^{PBTL} ::= P^{PLTL} \mid \mathbf{A}(P^{PBTL})$$

  We also use the following abbreviation for each $p \in \mathcal{G}^{PBTL}(\Delta)$:

  **(D11-E)** $\mathbf{E}p \overset{\text{def}}{=} \neg\mathbf{A}(\neg p)$.

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{PBTL}$, the entailment relation $\vdash_{\Delta}^{PBTL}$ is generated by the proof calculus of $PLTL$ together with the following one, provided that they are both stated over $\mathcal{G}^{PBTL}(\Delta)$, wherein $p$ and $q$ are included:

  **(A13-A)** $\Vdash_{\Delta}^{PBTL} \mathbf{A}(p \to q) \to (\mathbf{A}p \to \mathbf{A}q)$;

  **(A14-A)** $\Vdash_{\Delta}^{PBTL} \mathbf{A}p \to p$;

  **(A15-EA)** $\Vdash_{\Delta}^{PBTL} \mathbf{E}p \to \mathbf{AE}p$;

  **(A16-EV)** $\Vdash_{\Delta}^{PBTL} (\mathbf{E}p)\mathbf{V}q \to \mathbf{E}(p\mathbf{V}q)$;

  **(A17-AXU)** $\Vdash_{\Delta}^{PBTL} \mathbf{A}(p \to \mathbf{X}(q\mathbf{U}p)) \to (p \to \mathbf{XA}(q\mathbf{U}p))$;

  **(A18-Ebeg)** $\Vdash_{\Delta}^{PBTL} \mathbf{E}(\mathbf{beg}) \to \mathbf{beg}$;

  **(R4-A)** $\{p\} \Vdash_{\Delta}^{PBTL} \mathbf{A}p$. □

This defines a full branching time modality, in the sense that there is no restriction on using **A** in the scope of any other connective. Interesting logics with a nesting restriction do exist such as the Computation Tree Logic ($CTL$) of Emerson (1990). Also note that Lamport (1994), although considering an axiomatisation of linear time only, include in his logic an *enabledness* connective $En$ analogous to our **E** without providing corresponding logical axiom schemas or inference rules to support its derivation.

Axiom schemas **A13-15** and rule **R4-A** for modal generalisation make of **A** an $S5$ modality, which is determined here as usual by an equivalence relation on the worlds that occur at the same level in the set of legal behaviours. This reduced set of schemas and rules is distinct from (although equivalent to) the usual $S5$ axiomatisation, which is defined using another set of schemas (Goldblatt 1992; Exercise 2.8). Axiom **A18** says that the possibility of the current moment being initial forces it to be the case, meaning that all behaviours are at first synchronised, i.e., the level of their initial worlds is the same. **A16** considers in addition that a behaviour possesses an alternative in a future moment only if its subsequent history up to but not including that point could also be realised by the alternative behaviour. Schema **A17** extends this requirement in a pointwise manner by including the current moment in each future history.

Taking into account the preceding set of axiom schemas, we see that what makes our branching time logic really different from other formalisms is the interpretation assigned to **E**. Here we read a formula $\mathbf{E}p$ as $p$ occurring in some possible behaviour with an identical past history not necessarily including the current moment (or world). This means that **E** has a strict interpretation here. This interpretation easily yields invalid an axiom schema for non-strictness proposed by Stirling (1992): $\mathbf{E}p \to p$ for any atomic $p$. As an advantage we obtain that the substitution property still holds, which means that it is possible to substitute formulas by logically equivalent ones in any sentence. This is, however, a temporary achievement since we loose this property when considering a first-order extension of this branching time logic. Our interpretation also entails for the same reason that the logic above is substantially different from $CTL^*$ (Emerson 1990), where the same connective refers to behaviours with an identical past history necessarily including the current world. For atomic $p$,

$$\mathbf{E}p \wedge \neg p \to \mathbf{X}p \tag{2.5.1}$$

has a contradictory antecedent in $CTL^*$ whereas this need not be the case here. We do not know how to express in $CTL^*$ this property that $p$ happens at most each other moment in any behaviour. On the other hand, considering a definition of that logic in terms of our syntax[10], the theories of $CTL^*$ are interpreted into $PBTL$ by a functor $\mathcal{B} : \mathbf{Th}^{CTL^*} \to \mathbf{Th}^{PBTL}$ mapping signatures as the identity and each $p \in Th_\Delta(\Psi)$, $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{CTL^*}$ and $\Psi \subseteq \mathcal{G}(\Delta)$, into $\mathbf{beg} \to \mathcal{B}^*(p)$ belonging to $\mathcal{B}(Th_\Delta(\Psi))$, where $\mathcal{B}^*$ is defined as follows:

[10]Note that **beg** does not have a syntactic counterpart in $CTL^*$.

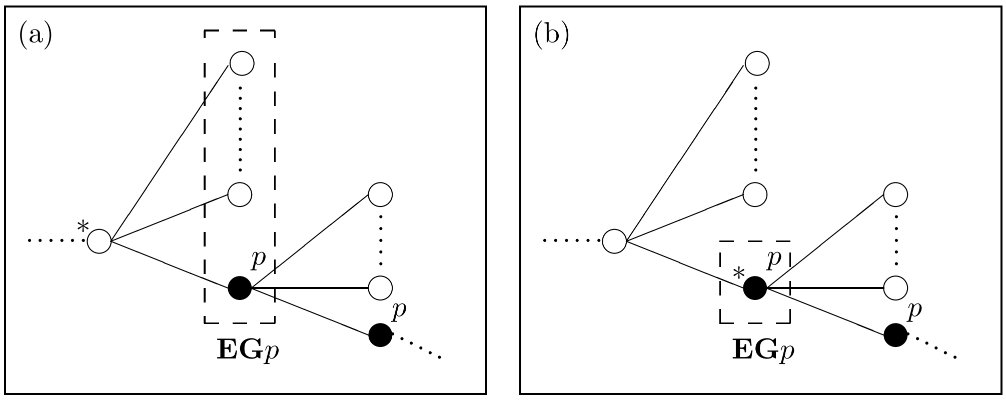


Figure 2.10: Interpretation of branching modality in (a) $PBTL$ and (b) $CTL^*$.

$$\begin{array}{ll}
\mathcal{B}^*(p) & \stackrel{\text{def}}{=} p, \text{ for } p \in \mathcal{E}^{CTL^*}(\Delta) \\
\mathcal{B}^*(\neg p) & \stackrel{\text{def}}{=} \neg \mathcal{B}^*(p) \\
\mathcal{B}^*(p \to q) & \stackrel{\text{def}}{=} \mathcal{B}^*(p) \to \mathcal{B}^*(q) \\
\mathcal{B}^*(p\mathbf{V}q) & \stackrel{\text{def}}{=} (\mathcal{B}^*(p))\mathbf{V}(\mathcal{B}^*(q)) \\
\mathcal{B}^*(\mathbf{A}p) & \stackrel{\text{def}}{=} p, \text{ for } p \in \mathcal{E}^{CTL^*}(\Delta) \\
\mathcal{B}^*(\mathbf{A}(p\mathbf{V}q)) & \stackrel{\text{def}}{=} \mathbf{XA}((\mathcal{B}^*(q))\mathbf{U}(\mathcal{B}^*(p))) \\
\mathcal{B}^*(\mathbf{A}p) & \stackrel{\text{def}}{=} \mathbf{A}(\mathcal{B}^*(p)), \text{ for any other } p
\end{array}$$

and each theory morphism $\psi : Th_{\Delta_1}(\Psi_1) \to Th_{\Delta_2}(\Psi_2)$ in $\mathsf{morph}\ \mathbf{Th}^{CTL^*}$ into $\mathcal{B}(\psi) : \{\mathcal{B}(p) | p \in Th_{\Delta_1}(\Psi_1)\} \to \{\mathcal{B}(p') | p' \in Th_{\Delta_2}(\Psi_2)\}$. Zanardo (1996) also studies many similar temporal logics.

Branching time logics are regarded as particular many dimensional formalisms by Gabbay *et al.* (1994). Essentially, new dimensions require additional arguments in interpreting each formula. We use trees as a conceptual abstraction of parallel behaviours in Figure 2.10 to clarify this interpretation. We indicate therein points of reference for interpretation using $(*)$ and sets of possible evaluation points of a formula are circumscribed by dashed boxes. In $CTL^*$, evaluation and reference points coincide as shown in Figure 2.10 (b). This is not, however, the requirement in our case. Hence, any behaviour passing through $(*)$ in Figure 2.10 (a) makes a formula $\mathbf{EG}p$ true because there is some future history following that past moment satisfying $\mathbf{G}p$.

It is customary to impose additional restrictions on the flows of time in bidimensional logics as a means of capturing the behaviour of computer programs in a more realistic manner (Emerson 1983). Such restrictions are:

**Prefix closure:** Prefixing transitions to a behaviour results in a valid behaviour;

**Suffix closure:** Every suffix of a behaviour is itself a valid behaviour;

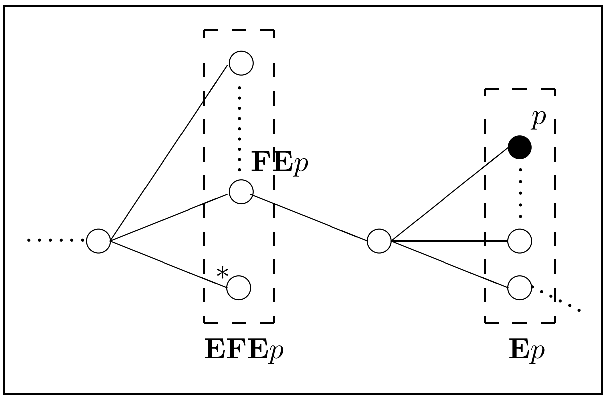


Figure 2.11: $\mathbf{FEFE}p \rightarrow \mathbf{FE}p$ is not valid in $PBTL$.

**Fusion closure:** Joining the past and the future of distinct behaviours at a shared moment always results in another valid behaviour;

**Limit closure:** If a behaviour can be followed for an arbitrarily long but finite length of time, it can be followed for an infinite length of time.

Suffix closure is invalid here since we adopt initialised behaviours and the process of taking out their initial segments does not guarantee that the remaining histories have acceptable initial moments, as identified by Manna and Pnueli (1989). Prefix closure is also invalid since finite behaviours are not admissible. Fusion closure is not supported as well because we can write, based on the initial moment, sentences that can distinguish two future histories sharing some moment, although Stirling (1992) mentions that this property may be captured by $\mathbf{AX}p \rightarrow \mathbf{XA}p$ and this schema is derivable from **A16**. Finally, we do not assume limit closure because doing so would prevent us from treating important notions of fairness. In fact, Emerson (1983) recognises that these assumptions do not always make sense, specially in representing real life objects or computational processes which have a definite notion of state. All the axiom schemas above are solely derived from the interpretation given to our branching modality.

Once again, the supermarket example can be used to illustrate the application of our logical system. Taking as a starting point the specifications of last section, let us assume that each component can provide some visual information on its current state. The locker, for instance, is able to signal that the trolley has been removed. The additional action symbols representing the display of visual information are introduced in Figure 2.12. Suppose, moreover, that, due to physical limitations, it is still impossible for each of these objects to allow the occurrence of more than one action at each time. We capture this constraint

**Specification** LOCKERPBTL
**attributes** $trolley\_in$
**actions** *release*, *lock*, *show_out*
**axioms**

$$release \vee lock \rightarrow \neg show\_out \tag{8.1}$$
$$show\_out \rightarrow \neg trolley\_in \tag{8.2}$$
$$\neg trolley\_in \rightarrow \mathbf{E}(show\_out) \tag{8.3}$$
$$\neg lock \wedge \mathbf{E}(show\_out) \rightarrow Just(show\_out) \tag{8.4}$$

**End**

**Specification** SAFEPBTL
**attributes** $card\_in$
**actions** *deposit*, *collect*, *show_in*
**axioms**

$$deposit \vee collect \rightarrow \neg show\_in \tag{9.1}$$
$$show\_in \rightarrow card\_in \tag{9.2}$$
$$card\_in \rightarrow \mathbf{E}(show\_in) \tag{9.3}$$
$$\neg collect \wedge \mathbf{E}(show\_in) \rightarrow Just(show\_in) \tag{9.4}$$

**End**

Figure 2.12: Specification of the supermarket system in $PBTL$.

through axioms (8.1) and (9.1). This means that a safe connected to the system as shown in Figure 2.13 is prevented from allowing *show_out* to happen whenever *lock* or *release* occur, which are in turn mutually exclusive actions because of (2.1). Since we do not want to preclude customers from performing any action, we cannot require that state information be displayed due to this disjointedness constraint. Instead, we adopt axioms (8.3) and (9.3) stating that information may be presented whenever each object is in use. These are so-called *willingness* properties (Barreiro *et al.* 1995), which say that each object is willing to perform an action although its occurrence is not guaranteed.

Willingness properties alone are too weak to force the occurrence of any action. In our example, although endowed with enough structure to display their busy state, there may be behaviours in which both safe and locker are being used, but never present such information. Then, the additional structure we have just defined would be useless. To increase the effectiveness of the system, we can enforce some weak fairness conditions, defined using the following abbreviation:

$$Just(p) \stackrel{\text{def}}{=} \mathbf{F}(p \vee \mathbf{A}(\neg p))$$

Axiom (8.4) specifies that whenever it is possible for the locker to display some information and the trolley is not being returned, its state will be presented in

$$\begin{tikzcd}
 & \textsc{Locker}\text{CP+L+BTL} \arrow[rr, "\overline{\sigma}_1^{\#}"'] & & \textsc{System}\text{PBTL} & & \textsc{Safe}\text{CP+L+BTL} \arrow[ll, "\overline{\sigma}_2^{\#}"] & \\
\textsc{Locker}\text{PBTL} \arrow[ur, "\sigma_{1.1.1}^{\#}"] & & \textsc{Locker}\text{CP+LTL} \arrow[ul, "\sigma_{1.1.2}^{\#}"'] & & \textsc{Safe}\text{CP+LTL} \arrow[ur, "\sigma_{2.2.1}^{\#}"] & & \textsc{Safe}\text{PBTL} \arrow[ul, "\sigma_{2.2.2}^{\#}"'] \\
 & \textsc{Locker} \arrow[ul, "\tau_{1.1.1}^{\#}"] \arrow[ur, "(\sigma_{1.2}\circ\tau_{1.2})^{\#}"] & & \textsc{Device} \arrow[ul, "(\sigma_{1.2}\circ\tau_{1})^{\#}"'] \arrow[ur, "(\sigma_{2.1}\circ\tau_{2})^{\#}"'] & & \textsc{Safe} \arrow[ul, "(\sigma_{2.1}\circ\tau_{2.1})^{\#}"'] \arrow[ur, "\tau_{2.2.2}^{\#}"'] &
\end{tikzcd}$$

Figure 2.13: Configuration of the supermarket system in $PBTL$.

the future (which possibly includes the current instant) or else this will become momentarily impossible for the locker. Because of (8.3), we can infer that in this last situation the trolley would have to be back again. On the other hand, for the trolley remaining in use indefinitely, the locker would eventually be obliged to present some information. Summarising these informal observations, we can now ensure that the display would work eventually whenever allowed to do so by the actions of supermarket customers (or possibly thieves).

Fairness axioms allow us to define in a modularised way how each object constrains the environment. A supermarket could not prevent information from being displayed, perhaps imposing additional conditions over the occurrence of *show_in*, while employing the locker and safe above. This would contradict our fairness axioms. We may, on the other hand, try to describe a perfect system, which would display state information whenever it had the opportunity. The following formula concerning the safe would be valid in this case:

$$\neg \mathit{collect} \wedge \mathbf{E}(\mathit{show\_in}) \rightarrow \mathit{show\_in} \tag{2.5.2}$$

Unfortunately, although a specification with formula (2.5.2) substituting (9.4) would not be inconsistent, the sequential perfect safe specified as a result would not have a display realisable as computer program: the decision as to whether or not to present some information would have to consume no time.

## 2.6 Classical First-Order Logic

Unless augmented with additional logical connectives to capture the peculiar characteristics of specific domains, an example being the temporal connectives defined in the previous sections to deal with time, propositional logics are unable to express in full generality the properties of individual objects in the context of their collections. For instance, in our previous example considering the supermarket system, it would be impossible to require from each customer to return the same released trolley because there were no logical means to say that any other trolley would not be acceptable. Much the same occurs with deposited identification cards. Another more problematic example related to computing is obtained by considering concurrent message passing systems, which cannot be faithfully specified within the realm of propositional logics because each exchanged message needs to be tagged in an unique way to ensure it will not be replicated. If we assume that time does not have end points and see that many messages may be dispatched at each instant, it is easy to conclude that the set of tags needs to be infinite. But we cannot talk about the infinitary character of some domain with finite proposition symbols and finitary connectives only.

Indeed, the weakness of propositional logical systems appears to lie in the absence of linguistic means to pick a denotation of an object in each domain of discourse and relate it to the denotation of any other object therein. This justifies a shift to first-order formalisms, which, through the use of logical variables and quantifiers, allow us to deal with these issues. Ignoring for a while the time dimension, we define below what we understand by *classical first-order logic.* Once again, we face the definition of a well-established logic, already studied by a number of authors such as van Dalen (1994). We take advantage of this fact to introduce in what follows most of the notation to be used in the remainder of the thesis.

**Definition 2.6.1 (First-Order Logic)** The entailment system of *classical first-order logic*, $FOL$ for short, is defined as follows:

- $\mathbf{Sig}^{FOL} \cong \mathbf{FinSet} \times \mathbf{FinSet}$ such that there exist:

    1. $Pred, Funct : \mathbf{Sig}^{FOL} \to \mathbf{FinSet}$, forgetful functors assigning signatures to disjoint sets of *predicate* and *function* symbols, respectively;

    2. $Type$, a map assigning each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{FOL}$ to a *similarity type* $(arity^{Pred}_{\Delta}, arity^{Funct}_{\Delta})$, each $arity$ itself a function with $\mathsf{dom}\ arity^{Pred}_{\Delta} \stackrel{\text{def}}{=} Pred(\Delta)$, $\mathsf{dom}\ arity^{Funct}_{\Delta} \stackrel{\text{def}}{=} Funct(\Delta)$ and $\mathsf{cod}\ arity^{Pred}_{\Delta} \stackrel{\text{def}}{=} \mathsf{cod}\ arity^{Funct}_{\Delta} \stackrel{\text{def}}{=} \mathbb{N}$. We usually drop the indexes from each $arity$;

- $\mathcal{L}^{FOL} \stackrel{\text{def}}{=} \mathcal{L}^{CPL} \cup \{\forall, \cdot\} \cup \mathcal{V}^{FOL}$ (a set of *variables*), such that $|\mathcal{V}^{FOL}| \stackrel{\text{def}}{=} \aleph_0$;

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{FOL}$, $\mathcal{E}^{FOL}(\Delta) \stackrel{\text{def}}{=} Pred(\Delta) \cup Funct(\Delta)$;

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{FOL}$, $Term(\Delta)$, $Atom(\Delta)$ and $\mathcal{G}^{FOL}(\Delta)$ are respectively sets of *terms*, *atomic formulas* and *formulas* defined by $T^{FOL}$, $A^{FOL}$ and $F^{FOL}$ as follows, provided that $x \in \mathcal{V}^{FOL}$, $f \in Funct(\Delta)$ with $arity(f) = m$ and $a \in Pred(\Delta)$ with $arity(a) = n$:

$$\begin{array}{l}
T^{FOL} ::= x \mid f(T_1^{FOL}, \ldots, T_m^{FOL}) \\
A^{FOL} ::= a(T_1^{FOL}, \ldots, T_n^{FOL}) \\
F^{FOL} ::= A^{FOL} \mid \neg F^{FOL} \mid F^{FOL} \rightarrow F^{FOL} \mid (F^{FOL}) \mid \forall x \cdot F^{FOL}
\end{array}$$

We also define a functor $Expr : \mathbf{Sig}^{FOL} \rightarrow \mathbf{Set}$ associating $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{FOL}$ to a set of *expressions* $Expr(\Delta) \stackrel{\text{def}}{=} \mathcal{G}^{FOL}(\Delta) \cup Term(\Delta) \cup \mathcal{E}^{FOL}(\Delta)$.

$FOL$ is equipped with a map $Free$ which assigns each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}$ to a *free variable* function $Free_\Delta : Expr(\Delta) \rightarrow \mathcal{P}(\mathcal{V}^{FOL})$. For $x \in \mathcal{V}^{FOL}$, $t_i \in Term(\Delta)$ and $\{p, p_i\} \subset Expr(\Delta)$, $Free(p)$ is defined as follows:

$$\begin{array}{ll}
Free(x) & \stackrel{\text{def}}{=} \{x\} \\
Free(f(t_1, \ldots, t_n)) & \stackrel{\text{def}}{=} \bigcup\{Free(t_i) | 1 \leq i \leq n\}, \\
 & \text{for } f \in Funct(\Delta) \text{ and } arity(f) = n \\
Free(a(t_1, \ldots, t_n)) & \stackrel{\text{def}}{=} \bigcup\{Free(t_i) | 1 \leq i \leq n\}, \\
 & \text{for } a \in Pred(\Delta) \text{ and } arity(a) = n \\
Free(\neg p) & \stackrel{\text{def}}{=} Free(p) \\
Free(p_1 \rightarrow p_2) & \stackrel{\text{def}}{=} Free(p_1) \cup Free(p_2) \\
Free(\forall x \cdot p) & \stackrel{\text{def}}{=} Free(p) - \{x\} \\
Free(p) & \stackrel{\text{def}}{=} \{\}, \\
 & \text{for any other } p \in Expr(\Delta)
\end{array}$$

To stress that $\{x, y\} \subseteq Free(p)$ must be the case, we write $p[x, y]$.

$FOL$ is also equipped with a map $[\cdot]$ associating each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{FOL}$ to a *substitution function* $[\cdot]_\Delta : \mathcal{G}^{FOL}(\Delta) \times \mathcal{V}^{FOL} \times Term(\Delta) \rightarrow \mathcal{G}^{FOL}(\Delta)$. For any $\{t_i, r\} \subseteq Term(\Delta)$, $q \in \mathcal{V}^{FOL}$ and $p \in \mathcal{G}^{FOL}(\Delta)$, $p[q\backslash r]$ denoting the substitution of $q$ by $r$ in $p$ is defined as follows:

$$\begin{array}{ll}
\underline{q}[q\backslash r] & \stackrel{\text{def}}{=} q \\
\underline{f(t_1, \ldots, t_n)}[q\backslash r] & \stackrel{\text{def}}{=} f(t_1[q\backslash r], \ldots, t_n[q\backslash r]), \\
 & \text{for } f \in Funct(\Delta) \text{ and } arity(f) = n \\
\underline{a(t_1, \ldots, t_n)}[q\backslash r] & \stackrel{\text{def}}{=} a(t_1[q\backslash r], \ldots, t_n[q\backslash r]), \\
 & \text{for } a \in Pred(\Delta) \text{ and } arity(a) = n \\
\underline{(\neg p_1)}[q\backslash r] & \stackrel{\text{def}}{=} \neg p_1[q\backslash r] \\
\underline{(p_1 \rightarrow p_2)}[q\backslash r] & \stackrel{\text{def}}{=} p_1[q\backslash r] \rightarrow p_2[q\backslash r] \\
\underline{(\forall x \cdot p_1)}[q\backslash r] & \stackrel{\text{def}}{=} \forall x \cdot (p_1[q\backslash r]), \\
 & \text{for } x \in \mathcal{V}^{FOL} - Free(r) \\
p[q\backslash r] & \stackrel{\text{def}}{=} p, \\
 & \text{for any other } p,\ q \text{ and } r.
\end{array}$$

where $p$ is the underlined expression in each case above.

We say that $r$ is free for $q$ in $p$ if and only if each occurrence of $q$ in $p$ does not appear in the scope of a quantifier which binds some of the free variables of $q$ in common with $r$. We only consider a substitution $p[q\backslash r]$ to be admissible if $r$ is free for $q$ in $p$. We also assume the existence of a *substitution relation* $\{\cdot\}$ associated to $[\cdot]$ which performs just some of the specified substitutions.

The following abbreviation is used for each $p \in \mathcal{G}^{FOL}(\Delta)$ and $x \in \mathcal{V}^{FOL}$:

**(D12-∃)** $\exists x \cdot p \overset{\text{def}}{=} \neg\forall x \cdot \neg p$.

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{FOL}$, the entailment relation $\vdash^{FOL}_{\Delta}$ is generated by the proof calculus of $CPL$ together with the following one, provided that they are both stated over $\mathcal{G}^{FOL}(\Delta)$, wherein $p$ and $q$ are included, that $x \in \mathcal{V}^{FOL}$ with $x \notin Free(p)$ and that $t \in Term(\Delta)$ is free for $x$ in $p$:

  **(A19-∀)** $\Vdash^{FOL}_{\Delta} (\forall x \cdot p[x]) \rightarrow p[x\backslash t]$;

  **(A20-∀)** $\Vdash^{FOL}_{\Delta} \forall x \cdot (p \rightarrow q) \rightarrow (p \rightarrow \forall x \cdot q)$;

  **(R5-∀)** $\{p \rightarrow q\} \Vdash^{FOL}_{\Delta} p \rightarrow \forall x \cdot q$. □

The definition of first-order logic is more elaborated than the propositional case. The category of signatures is endowed with linguistic structure to represent generic properties of elements using predicate symbols and functional relationships between them through function symbols. These elements are denoted by arguments in applying such symbols as well as the result in the case of functions. This is why first-order logic symbols are assigned to an arity, to define the number of elements involved in these situations. The logical language also contains a countably infinite set of variables and a quantifier symbol which allow us to express properties in generic form. These notions are standard.

What is unusual in our definition above is the use of a relation instead of a function to deal with substitution. We shall see in what follows that this additional generality is required in stating the properties of logical equality. A similar notion of *parallel substitution* is proposed by van Dalen (1994), who considers only substitutions of terms for variables within formulas but allows many of them to be effected in parallel producing non-deterministic results, which do not necessarily denote a single formula.

$FOL$ is a faithful extension of $CPL$. Indeed, all the additional axiom schemas and rule above have only to do with the newly introduced quantifier. **A19** says that properties of particular elements follow from the general case covering all the elements of the domain. Moreover, **A20** says that if a generic property guarantees for each element of the domain another property, so does it guarantee that the whole domain enjoys the same. **R5** is the universal generalisation rule. To see that this is a faithful extension of $CPL$, first note that there is a functor $\mathcal{F} : \mathbf{Sig}^{CPL} \to \mathbf{Sig}^{FOL}$ such that, for each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{CPL}$, $Funct(\mathcal{F}(\Delta))$ is empty and each $p \in \mathcal{E}^{CPL}(\Delta)$ is isomorphically mapped into $\mathcal{F}(p) \in Pred(\mathcal{F}(\Delta))$ with $arity(\mathcal{F}(p)) = 0$, and that also maps propositional morphisms accordingly. Compositional application to the symbols in each expression lifts $\mathcal{F}$ to another functor between the respective categories of theories. Each theory $Th_\Delta(\Psi)$ in $\mathsf{obj}\ \mathbf{Th}^{CPL}$ is interpreted into first-order logic because none of the theorems in $Th_\Delta(\Psi)$ is lost in the translation. This interpretation is also faithful because, when we restrict the language of a first-order theory to the co-domain of $\mathcal{F}$, it is necessarily the image of a theory in $CPL$. Note that $CPL$ is faithfully embedded into $PLTL$, which in turn is similarly embedded into $PBTL$, but in those cases the embedding functors are trivial.

Applying the preceding functor to the theories specified in Section 2.3, we obtain a set of examples of first-order theories. In examples requiring the full expressiveness of the logic, we assume that free variables in axioms are implicitly universally quantified. In practice, however, $FOL$ does not seem to be adequate to support the design of extensible systems. We need equality to deal with identity and the temporal connectives to recover direct access to time without resorting to any form of coding. Because of these reasons, we shall postpone the presentation of additional examples to Chapter 3.

### 2.6.1 Many-Sorted Logic with Equality

The examples we have provided point to the fact that the real world can be organised in collections of similar objects. Identification card numbers, message tags and others are instances of this idea. To be effective, this classification of the universe in domains requires additional support for defining similarity and sameness. These can be treated within a logic where the notions of sort and equality are made a logical part of the formalism, as defined below:

**Definition 2.6.2 (Many Sorted First-Order Logic)** The entailment system of *many sorted first-order logic with equality*, $MSFOL$, is defined as follows:

- $\mathbf{Sig}^{MSFOL} \cong \mathbf{Sig}^{FOL} \times \mathbf{FinSet}$ such that there exist:

    1. $Pred$ and $Funct$ as defined in $FOL$, and $Sort : \mathbf{Sig}^{MSFOL} \to \mathbf{FinSet}$ assigning each signature to a set of *sort* symbols disjoint from those of predicates and functions;

2. *Type*, a map assigning each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSFOL}$ to a *type signature* $(type_\Delta^{Pred}, type_\Delta^{Funct})$, each *type* itself a function with $\mathsf{dom}\ type_\Delta^{Pred} \stackrel{\text{def}}{=} Pred(\Delta)$, $\mathsf{dom}\ type_\Delta^{Funct} \stackrel{\text{def}}{=} Funct(\Delta)$ and $\mathsf{cod}\ type_\Delta^{Pred} \stackrel{\text{def}}{=} Sort(\Delta)^*_{fin}$, $\mathsf{cod}\ type_\Delta^{Funct} \stackrel{\text{def}}{=} Sort(\Delta)^*_{fin} \to Sort(\Delta)$. We usually drop the indexes from each *type* and put $arity(p) \stackrel{\text{def}}{=} \mathsf{len}\ (\mathsf{dom}\ type(p))$;

- $\mathcal{L}^{MSFOL} \cong \mathcal{L}^{FOL} \cup \{=\}$ such that there is a functor $Class : \mathbf{Sig}^{MSFOL} \to \mathbf{Set}^{\overset{\circ}{\to}}$ assigning each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSFOL}$ to a partial *classification* function $Class(\Delta) : \mathcal{V}^{MSFOL} \to Sort(\Delta)$. For $s \in Sort(\Delta)$, $\mathcal{V}_{\Delta_s}^{MSFOL} \stackrel{\text{def}}{=} \{x \in \mathcal{V}^{MSFOL} | Class(\Delta)(x) = s\}$, the set of $s$-classified variables;

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSFOL}$, $\mathcal{E}^{MSFOL}(\Delta) \stackrel{\text{def}}{=} Pred(\Delta) \cup Funct(\Delta)$;

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSFOL}$ and $s \in Sort(\Delta)$, we write as $Term(\Delta)_s$ the set of $s$-classified terms defined as follows:

$$\{t \in Term(\Delta) | t \in \mathcal{V}_{\Delta_s}^{MSFOL} \vee \mathsf{cod}\ type(t) = s\}$$

Moreover, $Atom(\Delta)$ is redefined as follows:

$$A^{MSFOL} ::= A^{FOL} \mid T_s^{FOL} = T_s^{FOL}$$

The following conditions are respectively added to the definition of *Free* and $[\cdot]$ for *FOL*, providing $\{t_1, t_2\} \subset Term(\Delta)_s$ for some $s \in Sort(\Delta)$:

$$Free(t_1 = t_2) \stackrel{\text{def}}{=} Free(t_1) \cup Free(t_2)$$
$$\underline{(t_1 = t_2)}[q \backslash r] \stackrel{\text{def}}{=} t_1[q \backslash r] = t_2[q \backslash r]$$

We also use the following abbreviations:

**(D13-NEQ)** $t_1 \neq t_2 \stackrel{\text{def}}{=} \neg(t_1 = t_2)$;

**(D14-UNI)** $\exists!\, x \cdot p[x] \stackrel{\text{def}}{=} \exists x \cdot (p[x] \wedge \forall y \cdot p[y] \to x = y)$.

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSFOL}$, the entailment relation $\vdash_\Delta^{MSFOL}$ is generated by the proof calculus of *FOL* together with the following one, provided that they are both stated over $\mathcal{G}^{MSFOL}(\Delta)$, that $\{t, t_1, t_2\} \subset Term(\Delta)_s$ for some $s \in Sort(\Delta)$, that $p \in Atom(\Delta)$ and that $q \in Expr(\Delta)$:

**(A21-EQ)** $\Vdash_\Delta^{MSFOL} t = t$;

**(A22-EQ)** $\Vdash_\Delta^{MSFOL} t_1 = t_2 \to (p\{q \backslash t_1\} \to p\{q \backslash t_2\})$. $\square$

That is, the similarity type of first-order signatures is extended with typing information based on sets of extra-logical sort symbols. Terms and variables are classified accordingly. Equality is included as a logical symbol, for which the axiomatisation above is standard. The only exception is perhaps the use of the substitution relation in **A22** to allow the proof of theorems like $x = y \to (a(x,y) \to a(y,x))$, $a \in \mathit{Pred}(\Delta)$, $\{x,y\} \subset \mathcal{V}^{MSFOL}$, which are not provable considering an axiomatisation based on the usual substitution function. Sernadas *et al.* (1995) adopts the same notion. Together with **A21**, which asserts that equality is reflexive, the other characteristic properties of equivalence relations, symmetry and transitivity, are provable as stated in Appendix I.

It is easy to see that $FOL$ can be faithfully embedded into $MSFOL$. More interestingly, we can also obtain a similar embedding in the opposite direction. Define a functor $\mathcal{M} : \mathbf{Sig}^{MSFOL} \to \mathbf{Sig}^{FOL}$ such that, for each signature $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSFOL}$, each symbol $p \in \mathcal{E}^{MSFOL}(\Delta) \cup \mathcal{V}^{MSFOL} \cup \{=\}$ is isomorphically mapped into an image in $\mathcal{E}^{FOL}(\mathcal{M}(\Delta))$ with the same arity. So:

- $x \in \mathcal{V}^{MSFOL} \Rightarrow \mathcal{M}(x) \in \mathcal{V}^{FOL}$;
- $f \in \mathit{Funct}(\Delta) \Rightarrow \mathcal{M}(f) \in \mathit{Funct}(\mathcal{M}(\Delta)) \wedge \mathit{arity}(\mathcal{M}(f)) = \mathit{arity}(f)$;
- $a \in \mathit{Pred}(\Delta) \Rightarrow \mathcal{M}(a) \in \mathit{Pred}(\mathcal{M}(\Delta)) \wedge \mathit{arity}(\mathcal{M}(a)) = \mathit{arity}(a)$;
- $s \in \mathit{Sort}(\Delta) \Rightarrow \mathcal{M}(s) \in \mathit{Pred}(\mathcal{M}(\Delta)) \wedge \mathit{arity}(\mathcal{M}(s)) = 1$;
- $\mathcal{M}(=) \in \mathit{Pred}(\mathcal{M}(\Delta)) \wedge \mathit{arity}(\mathcal{M}(=)) = 2$.

$\mathcal{M}$ lifts to a functor between the respective categories of theories by compositional application to $MSFOL$ expressions obeying what follows, provided $\{p,q\} \subseteq \mathcal{G}^{MSFOL}(\Delta)$, $\{t_1,t_2\} \subset \mathit{Term}(\Delta)_s$ for $s \in \mathit{Sort}(\Delta)$ and $x \in \mathcal{V}^{MSFOL}$:

$$\begin{array}{ll}
\mathcal{M}(t_1 = t_2) & \overset{\text{def}}{=} \mathcal{M}(=)(\mathcal{M}(t_1), \mathcal{M}(t_2)) \\
\mathcal{M}(\neg p) & \overset{\text{def}}{=} \neg \mathcal{M}(p) \\
\mathcal{M}(p \to q) & \overset{\text{def}}{=} \mathcal{M}(p) \to \mathcal{M}(q) \\
\mathcal{M}(\forall x \cdot p) & \overset{\text{def}}{=} \forall \mathcal{M}(x) \cdot \mathcal{M}(\mathit{Class}(\Delta)(x))(\mathcal{M}(x)) \to \mathcal{M}(p)
\end{array}$$

To see that this is also a faithful embedding, suppose that the restriction of a first-order theory to the language of the codomain of $\mathcal{M}$ contains a sentence which is not in the image of any $MSFOL$ theory. This is a contradiction since we know that any such $FOL$ theory can be faithfully embedded into $MSFOL$. The existence of both faithful embeddings means that these logics are equally expressive. So, what is the reason for introducing many-sorted logic with equality?

Sorts are a widely recognised way of making sentences more readable (van Dalen 1994). The justification of logical equality is more subtle and has to do with **A22** and alternative schemas. Assume our interest in specifying a problem involving an identity relation and at least one binary predicate symbol. To represent the characteristic properties of the relation is easy both in $FOL$ and $MSFOL$: in the first case they can be captured through three universally quantified axioms and nothing is needed in the second case by adopting the logical equality. On the other hand, to capture the substitution instances generated by the identity may be more demanding. Again, this is no difficulty for many-sorted logic with equality as being supported by the aforementioned schema. However, in the case of $FOL$ we would need to include infinitely many axioms in the specification. This is due to the impossibility of relying on equivalent terms to make substitutions (recall that replacement rules are derivable in both logical systems but they demand logically equivalent formulas as premises). Therefore, the problem is finitely axiomatizable in $MSFOL$ but not in $FOL$, meaning that it cannot be represented by a specification in the sense adopted here. For the sake of generality, we prefer the former logic.

It appears to be important to mention that the logic above, as an extension of unsorted classical first-order logic, does not suffer from the pathological anomaly of the similar extension based on equational logic, namely the unsoundedness of the extension identified by Goguen and Meseguer (1981). The anomaly appears in many-sorted equational logic because sort symbols denoting empty sets are allowed. It is easy to see that this is not the case in classical many-sorted logic as a consequence of the following theorem:

**Theorem 2.6.3 (Total terms)** Given a signature $\Delta$ in obj $\mathbf{Sig}^{MSFOL}$, the axiom schema below for any $t \in Term(\Delta)$ is provable in $MSFOL$:

**(NVOID)** $\Vdash_{\Delta}^{MSFOL} \exists y \cdot t = y$ (*terms do not have a partial interpretation*).

Proof:

1. $t = t$ **A21-EQ**
2. $\neg\neg\neg(t = t) \to \neg(t = t)$ **DOUB**
3. $(\neg\neg\neg(t = t) \to \neg(t = t)) \to (t = t \to \neg\neg(t = t))$ **A3-N**
4. $t = t \to \neg\neg(t = t)$ **R1-MP** 2, 3
5. $\neg\neg(t = t)$ **R1-MP** 1, 4
6. $\forall y \cdot \neg(t = y) \to \neg(t = t)$ **A19-∀**
7. $(\forall y \cdot \neg(t = y) \to \neg(t = t)) \to (\neg\neg(t = t) \to \neg\forall y \cdot \neg(t = y))$ **CONP**
8. $\neg\neg(t = t) \to \neg\forall y \cdot \neg(t = y)$ **R1-MP** 6, 7
9. $\neg\forall y \cdot \neg(t = y)$ **R1-MP** 5, 8
10. $\exists y \cdot t = y$ **D12-∃** 9 ■ **(NVOID)**

## 2.7 First-Order Temporal Logic

We have finally reached a point where it is possible to introduce a really expressive first-order temporal logical system to support software specification and verification. To assess the power of such a formalism in practice, it suffices to mention that it can support the representation of concurrent object systems with a variety of value-passing modes of interaction. It would appear straightforward to combine $PBTL$ and $MSFOL$ in a way which defines how the temporal connectives interact with the first-order quantifiers. However, such a combination presupposes many delicate decisions. We choose to define our first-order linear time entailment system as follows:

**Definition 2.7.1 (Many-Sorted Linear Time Logic)** The entailment system of *many-sorted linear time logic*, $MSLTL$, is defined as follows:

- $\mathbf{Sig}^{MSLTL} \cong \mathbf{Sig}^{MSFOL} \times \mathbf{FinSet}$ such that there exist:

  1. $Pred$, $Funct$ and $Sort$ as in $MSFOL$, with $Pred$ renamed as $Act$ (for *action* symbols), and there is an additional forgetful functor $Attr : \mathbf{Sig}^{MSLTL} \to \mathbf{FinSet}$ which assigns each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSLTL}$ to a set of *attribute* symbols, disjoint from those of actions, functions and sorts;
  2. *Type* as in $MSFOL$, assigning each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSLTL}$ to a type signature with a new component $type^{Attr}_{\Delta}$ with $\mathsf{dom}\ type^{Attr}_{\Delta} \stackrel{\text{def}}{=} Attr(\Delta)$ and $\mathsf{cod}\ type^{Attr}_{\Delta} \stackrel{\text{def}}{=} Sort(\Delta)^*_{fin} \to Sort(\Delta)$;

- $\mathcal{L}^{MSLTL} \cong \mathcal{L}^{MSFOL} \cup \mathcal{L}^{PLTL}$;

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSLTL}$, $\mathcal{E}^{MSLTL}(\Delta) \stackrel{\text{def}}{=} Act(\Delta) \cup Funct(\Delta) \cup Attr(\Delta)$;

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSLTL}$, $Term(\Delta)_s$ for $s \in Sort(\Delta)$ and $\mathcal{G}^{MSLTL}(\Delta)$ are (re)defined by $T^{MSLTL}_s$ and $F^{MSLTL}$ as follows, providing $x \in \mathcal{V}^{MSLTL}_{\Delta_s}$, $f \in Funct(\Delta)$ with $type(f) = \langle s_1, \ldots, s_m\rangle \to s$ and $g \in Attr(\Delta)$ with $type(g) = \langle s_1, \ldots, s_n\rangle \to s$:

$$\begin{aligned} T^{MSLTL}_s &::= x \mid f(T^{MSLTL}_{s_1}, \ldots, T^{MSLTL}_{s_m}) \mid g(T^{MSLTL}_{s_1}, \ldots, T^{MSLTL}_{s_n}) \\ F^{MSLTL} &::= F^{MSFOL} \mid \mathbf{beg} \mid (F^{MSLTL})\mathbf{V}(F^{MSLTL}) \end{aligned}$$

The following cases are added to the definition of $Free$ and $[\cdot]$ respectively:

$$\begin{array}{ll}
Free(g(t_1,\ldots,t_n)) \overset{\text{def}}{=} \bigcup\{Free(t_i)|1 \le i \le n\}\,, & \\
\qquad\qquad\qquad\qquad \text{for } g \in Funct(\Delta) \text{ and } arity(g) = n & \\
Free(p_1\mathbf{V}p_2) \qquad \overset{\text{def}}{=} Free(p_1) \cup Free(p_2) &
\end{array}$$

$$\begin{array}{ll}
\underline{g(t_1,\ldots,t_n)}[q\backslash r] \overset{\text{def}}{=} g(t_1[q\backslash r],\ldots,t_n[q\backslash r]) & \\
\qquad\qquad\qquad \text{for } g \in Attr(\Delta) \text{ and } arity(g) = n & \\
\underline{(p_1\mathbf{V}p_2)}[q\backslash r] \qquad \overset{\text{def}}{=} (p_1[q\backslash r])\mathbf{V}(p_2[q\backslash r]) &
\end{array}$$

- For each $\Delta$ in obj $\mathbf{Sig}^{MSLTL}$, the entailment relation $\vdash_{\Delta}^{MSLTL}$ is generated by the proof calculi of $MSFOL$, $PLTL$ and the following one, provided that they are all stated over $\mathcal{G}^{MSLTL}(\Delta)$, wherein $p$ and $q$ are included, that $x \in \mathcal{V}^{MSLTL}$ with $x \notin Free(q)$ and $\{t_1, t_2\} \subset Term(\Delta)_s$ for some $s \in Sort(\Delta)$ are such that no attribute symbol appears in $t_i$, $1 \le i \le 2$:

  **(A23-∃V)** $\Vdash_{\Delta}^{MSLTL} (\exists x \cdot p)\mathbf{V}q \rightarrow \exists x \cdot p\mathbf{V}q$;

  **(A24-EQG)** $\Vdash_{\Delta}^{MSLTL} t_1 = t_2 \rightarrow \mathbf{G}(t_1 = t_2)$;

  **(A25-NEQG)** $\Vdash_{\Delta}^{MSLTL} t_1 \neq t_2 \rightarrow \mathbf{G}(t_1 \neq t_2)$. □

We consider that, while some of the symbols in each signature remain with the same *rigid* interpretation adopted in classical first-order logic, some others acquire a *flexible*, temporalised meaning. Sort and function symbols always have the same denotation regardless of the moment or the behaviour in which they are evaluated. Predicates, on the other hand, now called actions, are to be understood as representing the occurrence of instantaneous events. Note that the extra-logical, immediate character of actions herein differs fundamentally from that of $TLA$ (Lamport 1994), where actions are abbreviational definitions of transitional formulas. We also include in each signature an additional set of flexible function symbols, attributes, to represent instantaneous state. Families of state symbols with slightly distinct definitions appear in the literature as rigid constants (Andréka *et al.* 1995), attribute symbols with empty domain, and global variables (Manna and Pnueli 1983, Lamport 1994), symbols as in $\mathcal{V}^{MSLTL}$ with a temporalised interpretation.

Taking into account the possibility of having variables, sort, function and predicate symbols with rigid or flexible interpretation (note that in our case we have families of function symbols in both categories) or even absent in a logic, and considering moreover that it may be reasonable to prevent quantification over some families of variables, it is not too difficult to conclude that the number of conceivable logics obtained by combination of these cases is higher than 250.

Most of these are almost identical or uninteresting. In some other cases such as for flexible sorts the axiomatisation of the corresponding interpretations becomes overly complicated for practical application. Apart from the formalisms already proposed in the literature aiming at the design of concurrent systems, it may be worthwhile investigating a logic formulated with rigid and flexible predicates as well as functions, with the purpose of capturing in a more natural way with the additional rigid symbols the persistent schemas and static integrity constraints usually found in database system applications. We shall not explore this alternative formulation any further here.

The language of $MSLTL$ is such that terms, atoms and formulas are construed almost as in $MSFOL$. In particular, we do not adopt $\mathbf{X}t$ as a term in the way proposed by Manna and Pnueli (1983) and later generalised by Fiadeiro and Sernadas (1990) since it is not clear if the expressive power of the logic increases at all. The proposed axiom schemas capture the choices described above. **A23** is a Barcan formula saying that quantification domains do not vary with the passing of time. It is due to Mark Reynolds in this form, which entails the more conventional $\forall x \cdot \mathbf{G}(p) \to \mathbf{G}(\forall x \cdot p)$. Note its similarity with **A16**, although in that case the converse is not valid. **A24-5** say that terms which do not include attribute symbols are rigid. Because of the side condition in these schemas, we loose the substitutivity property which would allow us to substitute sentences by logically equivalent ones in any context. Although we have already introduced linguistic support to write frame axioms, which may require, for instance, that only the actions of an object change the value of its attributes, we postpone their definition until Chapter 3 where we shall study an object-based approach to extensible systems design.

The formalisation of the choices above concerning the interpretation of signature symbols can be carried forward in an analogous way to branching time as defined below:

**Definition 2.7.2 (Many-sorted Branching Time Logic)** The entailment system of *many-sorted branching time logic*, $MSBTL$, is defined as follows:

- $\mathbf{Sig}^{MSBTL} \cong \mathbf{Sig}^{MSLTL}$;
- $\mathcal{L}^{MSBTL} \cong \mathcal{L}^{MSLTL} \cup \mathcal{L}^{PBTL}$;
- $\mathcal{E}^{MSBTL} \cong \mathcal{E}^{MSLTL}$;
- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSBTL}$, $\mathcal{G}^{MSBTL}(\Delta)$ is defined by $F^{MSBTL}$ as follows:

$$F^{MSBTL} ::= F^{MSLTL} \mid \mathbf{A}(F^{MSBTL})$$

The following conditions are added to the definition of *Free* and $[\cdot]$, respectively, providing $p \in \mathcal{G}^{MSLTL}(\Delta)$:

$$Free(\mathbf{A}p) \stackrel{\text{def}}{=} Free(p)$$
$$\underline{(\mathbf{A}p_1)}[q\backslash r] \stackrel{\text{def}}{=} \mathbf{A}(p_1[q\backslash r])$$

- For each $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSBTL}$, the entailment relation $\vdash_\Delta^{MSBTL}$ is generated by the proof calculus of $MSLTL$ together with the following one, provided that they are both stated over $\mathcal{G}^{BTMSL}(\Delta)$, wherein $p$ is included, that $x \in \mathcal{V}^{MSBTL}$ and $\{t_1, t_2\} \subset Term(\Delta)_s$ for some $s \in Sort(\Delta)$ are such that no attribute symbol appears in $t_i$, $1 \leq i \leq 2$:

**(A26-∀A)** $\Vdash_\Delta^{MSBTL} \forall x \cdot \mathbf{A}p \rightarrow \mathbf{A}(\forall x \cdot p)$;

**(A27-EQA)** $\Vdash_\Delta^{MSBTL} t_1 = t_2 \rightarrow \mathbf{A}(t_1 = t_2)$. □

**A26** and **A27**, respectively, play the roles of **A23** and **A24-5** with respect to the less complex branching modality.

We are now in a more comfortable position to study the required additional reasoning principles to support the verification of liveness properties. As is well known, due to the fact that the set of safety properties is closed under intersection (Alpern and Schneider 1985), it is not possible to verify a liveness property based only on a set of safety hypotheses. For this reason, liveness properties are usually stated as part of the axioms in each given specification or can be derived from particular fairness assumptions made in the axiomatisation of the temporal logic. In Chapter 3, we shall explore these possibilities to start the verification process.

It is also fundamental to be able to produce derivations of liveness properties from previously verified ones. The so-called lattice principle, introduced as a proof method by Owicki and Lamport (1982) and adopted as a basic inference rule in (Manna and Pnueli 1979, Lamport 1994), appears to be the most general way of supporting such derivations. Essentially, based on the premises that $\prec$ is a well-founded binary relation and that a property $p$ of a generic element $x$ implies either another distinguished property $q$ being obtained or another element $y$ related to $x$ being found with the property $p$, both facts related to the future, the rule allows one to infer that the existence of an element with the property $p$ implies the occurrence of the distinguished property $q$ in the future. Such an occurrence is ensured by the fact that there cannot be an infinitely decreasing chain of elements related by $\prec$, which is guaranteed by well-foundedness. In this way, at least two liveness properties are involved in the form of a complex premise and a simple conclusion. This inference rule can be stated as follows:

**(WELL)** $\{\forall x \cdot (p[x] \rightarrow \mathbf{F}(q \vee \exists y \cdot y \prec x \wedge p[y]))\} \Vdash_{\Phi}^{MSBTL} (\exists z \cdot p[z]) \rightarrow \mathbf{F}q.$

Two issues must be treated if such a reasoning principle is to be adopted: (i) to show how to specify and verify that some relation is well-founded; and (ii) to show that the inference rule above is admissible considering a particular logical system. The connection between well-founded orders and the principle of transfinite induction with respect to their axiomatisation for temporal reasoning was studied in detail by Kröger (1987). The requirement in his work of an order relation appears to be too strong in that transitivity is not necessary anywhere. In any case, an induction schema[11] remains which cannot be classically treated using the finitist methods for software development required here. This rules out the possibility of either specifying or verifying within classical first-order logic only that some formula defines a well-founded relation.

Abadi and Merz (1996) recently realised that the well-foundedness of a binary rigid relation may be axiomatised in some temporal logical systems. Using our own system, they would (pseudo)-axiomatise this property as follows:

**(IRR)** $\forall x \cdot \neg(x \prec x)$;

**(APROG)** $\mathbf{G}(\forall x \cdot t = x \rightarrow \mathbf{X}(t = x \vee t \prec x)) \rightarrow \mathbf{FG}(\forall x \cdot t = x \rightarrow \mathbf{X}(t = x))$

provided an arbitrarily chosen and unconstrained flexible symbol $t$ having the same sort as the relation. Intuitively, **IRR** says that $\prec$ is irreflexive. **APROG** relies on $t$ and the linear infinite discrete character of each behaviour to assess whether or not $\prec$ has an infinitely decreasing chain. If $t$ eventually becomes always invariant whenever it is bound to containing the values in a strictly decreasing chain, then such a chain necessarily has an end point since the value of the term could otherwise decrease forever in some behaviour. Because the same test is performed for every behaviour, since **APROG** implicitly encompasses any possible behaviour, and for every sequence of values for $t$, since this term is unconstrained, we can conclude that the relation is well-founded.

In order to hide the symbol $t$ and guarantee that it is unconstrained in **APROG**, Abadi and Merz (1996) adopted the quantification over flexible logical variables of $TLA$. Here, because our logic was intentionally made less expressive but simpler to define and use, we adopt constructions as follows:

**Definition 2.7.3 (Introduction of unconstrained flexible symbol)** Given a specification $\Phi_1 = (\Delta_1, \Psi_1)$ in obj $\mathbf{Pres}^{MSBTL}$ such that $s \in Sort(\Delta_1)$ and $\prec \in Act(\Delta)$, $type(\prec) = s \times s$, the specification $\Phi_2 = (\Delta_2, \Psi_2)$ in obj $\mathbf{Pres}^{MSBTL}$

[11] **(TI)** $\forall x \cdot ((\forall y \cdot y \prec x \rightarrow p[x \backslash y]) \rightarrow p[x]) \rightarrow \forall x \cdot p[x]$.

and the morphism $\Phi_1 \overset{\sigma^\#}{\to} \Phi_2$ in $\mathsf{morph}\ \mathbf{Pres}^{MSBTL}$ are an *extension of* $\Phi_1$ *by addition of unconstrained flexible symbol* $t$ if the following conditions are fulfilled:

- $t \in Attr(\Delta_2)$ with $type(t) = \epsilon \to \sigma(s)$ such that $t \notin \sigma(Attr(\Delta_1))$;
- $\Psi_2$ only contains $\sigma^\#(\Psi_1)$ and the following two axioms:

  **(FREE)** $\forall x \cdot \mathbf{E}(t = x)$;

  **(LIM)** $\mathbf{AG}(\forall x \cdot t = x \to \mathbf{XE}(t \prec x)) \to \mathbf{EG}(\forall x \cdot t = x \to \mathbf{X}(t \prec x))$. $\square$

**FREE** says that each element of the respective sort may at any moment be the value of the newly introduced flexible symbol $t$, ensuring in this way that $t$ is really unconstrained. **LIM** guarantees that, if there is always a possibility to follow an infinitely decreasing chain of elements related by $\prec$, then there is a behaviour in which $t$ follows the whole infinite chain, a limit closure axiom. The construction above can be regarded as a particular instance of the use of auxiliary symbols to support correctness proofs as originally proposed by Owicki and Gries (1976) to record part of the history of each behaviour. In particular, the auxiliary symbol $t$ is introduced here just to support the proof of well-foundedness but is not needed in (and is actually hidden from) the original specification and can be discharged afterwards. This is possible because the morphism in our definition can be shown to determine both a conservative extension and a model expansion, since the newly introduced properties are all concerning the symbol $t$. So, the extension does not really add new properties to the originally specified theory.

A careful reading of the literature on **WELL** shows that $\prec$ is assumed to be an extra-logical symbol with a given rigid interpretation. However, there is no reason for preventing the relation from being definable in terms of other symbols nor for disregarding changes in meaning as soon as well-foundedness is insured. More properties of software systems could be verified by weakening such assumptions. Therefore, it seems to be reasonable to propose a formula $r \in \mathcal{G}^{MSBTL}(\Delta)$, $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSBTL}$, to serve as a definition of $\prec$, i.e.:

$$x \prec y \leftrightarrow r[x, y]$$

such that $\prec$ does not appear in $r$ and $Free(r) = \{x, y\}$. This process can be made more systematic as in the introduction of unconstrained flexible symbols: a new specification is proposed containing in addition just the relation symbol and its defining axiom. The required morphism should be defined accordingly. This morphism is automatically made faithful in this way as representing an extension by explicit definition of a predicate like symbol. Note that $\prec \in Act(\Delta)$ must be the case because actions are the only relational symbols allowed in $MSBTL$ signatures. Whenever $r$ is written in terms of some flexible symbol, $\prec$ acquires a flexible meaning. Otherwise, $\prec$ is rigid and no change is required in the rationale above. It is important to stress that, should both extensions be necessary, the composition of the two involved morphisms may also result in another faithful morphism provided that the extension by definitions be carefully stated so as to prevent the symbol $t$ from appearing in the formula $r$. This would harm the correctness of the whole construction.

To assign a time-dependent meaning to $\prec$ and obtain a well-founded relation, we need to define formula $r$ in such a way that at least **IRR** and **APROG** are derivable. Clearly, such a definition cannot involve temporal connectives since the relation is supposed to associate elements of a particular domain at isolated time instants. This is called a state formula in the literature (Manna and Pnueli 1983). Now, even if the meaning of $\prec$ may change as time passes, **IRR** ensures that the relation is always anti-reflexive, which is fundamental because, if not guaranteed, it would be possible to witness the flexible symbol in **APROG** becoming permanently invariant even when all the decreasing chains of related elements are infinite, making **WELL** unsound. To admit some change without harming well-foundedness, we may require in addition that each change preserves currently related elements, a monotonicity requirement, and moreover that this process of change eventually stops, a termination requirement. These conditions prevent $\prec$ not only from having completely unrelated meanings in distinct moments but also from allowing decreasing chains which may be indefinitely extended by the addition of new elements. On the other hand, some originally unrelated elements may eventually leave this situation. Putting these requirements together, we reach the following axioms:

**(STAB)** $\forall x, y \cdot x \prec y \to \mathbf{X}(x \prec y)$

**(TERM)** $\mathbf{FG}(\forall x, y \cdot \neg(x \prec y) \to \mathbf{X}(\neg(x \prec y)))$

Let us deal with the second issue mentioned above, the admissibility of the lattice principle. It is not difficult to see that the inference rule **WELL** would be derivable if the following axiom schema were also derivable in $MSBTL$:

$$\forall x \cdot (\mathbf{F}(p[x]) \to \exists y \cdot \mathbf{F}(y \prec x \wedge p[y])) \to \forall z \cdot \neg\mathbf{F}(p[z]) \tag{2.7.1}$$

For a rigid relation symbol, this schema is equivalent to that of transfinite induction, which is known to lack a finite axiomatisation within classical first-order logical systems (Ryll-Nardzwski 1952). To see the equivalence, remove $\prec$ from the context of the second temporal connective above based on the assumption that this symbol is rigid and put $q[x] \overset{\text{def}}{=} \neg\mathbf{F}(p[x])$. The resulting sentence is equivalent to **TI**. Considering the flexible case, we could have some hope to show that (2.7.1) is derivable since first-order temporal logical systems such as $MSBTL$ which possess a linear infinite discrete time dimension are able to interpret minimal arithmetic (Abadi 1989) and thus mathematical induction can be made available. However, Gentzen (1943) proved that full transfinite induction is not derivable in any first-order arithmetical system. This is even true for some definitions of the standard ordering of the natural numbers as shown by Troelstra and Schwichtenberg (1996). As a consequence of these impossibility results, we could conclude that **WELL** is not derivable in any case.

Studying this situation, however, we can see that there are ways of overcoming the problem. Much in the way that temporal logic can be used to provide a (pseudo)-axiomatisation of well-foundedness, the same technique of extending the given specification with an unconstrained flexible symbol can be used to support an admissible proof rule having the schema above as the conclusion. So, because we can extend our logical system with such an admissible rule, the negative results mentioned above are not really restrictive. Gabbay (1981) uses the same idea of introducing new symbols in derivations in order to prove completeness of many propositional temporal logical systems. Here, the respective rule is stated as follows:

**Proposition 2.7.4 (Admissibility of INTRO)** Assume that $s \in \mathit{Sort}(\Delta)$, $t \in \mathit{Attr}(\Delta)$ with $\mathit{type}(t) = \epsilon \to s$ and $\prec\, \in \mathit{Act}(\Delta)$ with $\mathit{type}(\prec) = s \times s$ for a given $\Delta$ in obj $\mathbf{Sig}^{MSTBL}$. The following inference rule is admissible in $MSBTL$:

$$\textbf{(INTRO)}\quad \frac{\begin{array}{llll} 1. & \textbf{IRR} & 4. & \textbf{APROG} \\ 2. & \textbf{STAB} & 5. & \textbf{FREE} \\ 3. & \textbf{TERM} & 6. & \textbf{LIM} \end{array}}{\forall x \cdot (\mathbf{F}(p[x]) \to \exists y \cdot \mathbf{F}(y \prec x \land p[y])) \to \forall x \cdot \neg\mathbf{F}(p[x])}$$

Note that **INTRO**, a rule for introducing $t$ in derivations, has the axioms studied above as premises and (2.7.1) as the conclusion. If we can adopt this inference rule as part of our proof calculus, we can show that **WELL** is derivable. We postpone this admissibility proof until Section 2.9, calling the extended logical system $MSBTL^+$.

**Theorem 2.7.5 (Admissibility of WELL)** Assume that $s \in \mathit{Sort}(\Delta)$, $t \in \mathit{Attr}(\Delta)$ with $\mathit{type}(t) = \epsilon \to s$ and $\prec\, \in \mathit{Act}(\Delta)$ with $\mathit{type}(\prec) = s \times s$ for a given $\Phi = (\Delta, \Psi)$ in obj $\mathbf{Pres}^{MSTBL^+}$. Provided that **FREE**, **LIM**, **IRR**, **APROG**, **STAB** and **TERM** are derivable considering these symbols, the following inference rule, for $\{p, q\} \subseteq \mathcal{G}^{MSBTL^+}(\Delta)$ and $x \in \mathcal{V}^{MSBTL^+}$ such that $x \notin Free(q)$, is also derivable in $MSBTL^+$:

**(WELL)** $\{\forall x \cdot (p[x] \rightarrow \mathbf{F}(q \vee \exists y \cdot y \prec x \wedge p[y]))\} \Vdash_{\Phi}^{MSBTL^+} (\exists z \cdot p[z]) \rightarrow \mathbf{F}q.$

Proof: We follow the structure of the proof developed by Kröger (1987). Essentially, we need to substitute the use of the transfinite induction in his work by an application of the inference rule proposed above:

| | |
|---|---|
| 1. $\forall x \cdot (p[x] \rightarrow \mathbf{F}(q \vee \exists y \cdot y \prec x \wedge p[y])$ | **Ass** |
| 2. $\mathbf{G}(p[x] \rightarrow \mathbf{F}(q \vee \exists y \cdot y \prec x \wedge p[y]))$ | **A19-**$\forall$, **R1-MP** 1, **R2-G** |
| 3. $\mathbf{F}(p[x]) \rightarrow \mathbf{FF}(q \vee \exists y \cdot y \prec x \wedge p[y])$ | **MON-GF**, **R1-MP** 2 |
| 4. $\mathbf{FF}(q \vee \exists y \cdot y \prec x \wedge p[y]) \rightarrow \mathbf{F}(q \vee \exists y \cdot y \prec x \wedge p[y])$ | **IDEM-F** |
| 5. $\mathbf{F}(p[x]) \rightarrow \mathbf{F}(q \vee \exists y \cdot y \prec x \wedge p[y])$ | **HS** 3, 4 |
| 6. $\mathbf{F}(q \vee \exists y \cdot y \prec x \wedge p[y]) \rightarrow \mathbf{F}q \vee \mathbf{F}(\exists y \cdot y \prec x \wedge p[y])$ | **DIST-ORF** |
| 7. $\mathbf{F}(p[x]) \rightarrow \mathbf{F}q \vee \mathbf{F}(\exists y \cdot y \prec x \wedge p[y])$ | **HS** 5, 6 |
| 8. $\mathbf{F}(\exists y \cdot y \prec x \wedge p[y]) \rightarrow \mathbf{F}q \vee \exists y \cdot \mathbf{F}(y \prec x \wedge p[y])$ | **BARC-F**, **OR-R** |
| 9. $\mathbf{F}q \rightarrow \mathbf{F}q \vee \exists y \cdot \mathbf{F}(y \prec x \wedge p[y])$ | **REFL**, **OR-R** |
| 10. $\mathbf{F}q \vee \mathbf{F}(\exists y \cdot y \prec x \wedge p[y]) \rightarrow \mathbf{F}q \vee \exists y \cdot \mathbf{F}(y \prec x \wedge p[y])$ | **OR-L** 8, 9 |
| 11. $\forall x \cdot (\mathbf{F}(p[x]) \rightarrow \mathbf{F}q \vee \exists y \cdot \mathbf{F}(y \prec x \wedge p[y]))$ | **HS** 7, 10; **GEN-**$\forall$ |
| 12. $\mathbf{F}(p[z]) \rightarrow \mathbf{F}q$ | **INTRO**, **R1-MP** 11, **A19-**$\forall$, **R1-MP** |
| 13. $p[z] \rightarrow \mathbf{F}q$ | **REFL**, **OR-R**, **D8-F**, **HS** 12 |
| 14. $(\exists z \cdot p[z]) \rightarrow \mathbf{F}q$ | **GEN-**$\forall$ 13, **EXC-**$\forall\exists$, **HS** ■ **(WELL)** |

The application of **INTRO** is very demanding. We have to obtain beforehand all the special purpose axioms studied in this section. When the given relation is rigid, we can simplify this process by showing that **STAB** and **TERM** follow from the rigid interpretation of $\prec$. The theorem below allows us to generalise in time all the sentences written only in terms of rigid symbols:

**Theorem 2.7.6 (Invariant rigid formulas)** The axiom schema below is provable in $MSBTL$ for any sentence $p \in \mathcal{G}^{MSBTL}(\Delta) \cap S^+$, where $S$ is the set $S \stackrel{\text{def}}{=} \mathcal{V}^{MSBTL} \cup Funct(\Delta) \cup (\mathcal{L}^{MSFOL} - \{\mathbf{beg}\})$, for any $\Delta$ in obj $\mathbf{Sig}^{MSBTL}$:

**(RIGID)** $\Vdash_{\Delta}^{MSBTL} p \rightarrow \mathbf{G}p.$

Proof: We first examine atomic formulas and then proceed by structural induction on $\mathcal{G}^{MSBTL}(\Delta)$. Without attribute and action symbols in the underlying language, the possible atomic formulas can only be equality tests of the form $p \equiv (t_1 = t_2)$, for $\{t_1, t_2\} \subset Term(\Delta)_s$, $s \in Sort(\Delta)$. But we have **A24** which ensures $p \rightarrow \mathbf{G}p$ in this case. For the induction we have the following cases:

- $p \equiv \neg q$:

| | | |
|---|---|---|
| 1. | $(q \to \bot) \to \mathbf{G}(q \to \bot)$ | **Ind. Hyp.** |
| 2. | $\neg q \to (q \to \bot)$ | **NEG-L**, **PERM**, **R1-MP**, **D2-$\bot$** |
| 3. | $\neg q \to \mathbf{G}(q \to \bot)$ | **HS** 2, 1 |
| 4. | $(g \to \neg\bot) \to ((q \to \bot) \to \neg q)$ | **NEG-R**, **PERM**, **R1-MP** |
| 5. | $q \to \top$ | **A1-I**, **D1-$\top$** |
| 6. | $(q \to q) \to \neg\neg(q \to q)$ | **DOUB**, **A3-N**, **R1-MP** |
| 7. | $g \to \neg\bot$ | **D1-$\top$**, **D2-$\bot$** 6, **HS** 5 |
| 8. | $(q \to \bot) \to \neg q$ | **R1-MP** 4, 7 |
| 9. | $\mathbf{G}(q \to \bot) \to \mathbf{G}(\neg q)$ | **R2-G** 8, **MON-G**, **R1-MP** |
| 10. | $\neg q \to \mathbf{G}(\neg q)$ | **HS** 5, 9 |

- $p \equiv q \to r$:

| | | |
|---|---|---|
| 1. | $\neg q \to \mathbf{G}(\neg q)$ | **Ind. Hyp.** |
| 2. | $r \to \mathbf{G}r$ | **Ind. Hyp.** |
| 3. | $\neg q \vee r \to \mathbf{G}(\neg q) \vee \mathbf{G}r$ | **OR-R** 1, **OR-R** 2, **OR-L** |
| 4. | $\mathbf{G}(\neg q) \vee \mathbf{G}r \to \mathbf{G}(\neg q \vee r)$ | **DIST-ORG** |
| 5. | $\neg q \vee r \to \mathbf{G}(\neg q \vee r)$ | **HS** 3, 4 |
| 6. | $(\neg\neg q \to r) \to \mathbf{G}(\neg\neg q \to r)$ | **D3-OR** 5 |
| 7. | $(q \to r) \to (\neg\neg q \to r)$ | **DOUB**, **RTRAN**, **R1-MP** |
| 8. | $(q \to r) \to \mathbf{G}(\neg\neg q \to r)$ | **HS** 7, 6 |
| 9. | $q \to \neg\neg q$ | **DOUB**, **A3-N**, **R1-MP** |
| 10. | $(q \to \neg\neg q) \to ((\neg\neg q \to r) \to (q \to r))$ | **LTRAN** |
| 11. | $(\neg\neg q \to r) \to (q \to r)$ | **R1-MP** 9, 10 |
| 12. | $\mathbf{G}(\neg\neg q \to r) \to \mathbf{G}(q \to r)$ | **R2-G** 11, **MON-G**, **R1-MP** |
| 13. | $(q \to r) \to \mathbf{G}(q \to r)$ | **HS** 8, 12 |

- $p \equiv \forall x \cdot q$:

| | | |
|---|---|---|
| 1. | $q \to \mathbf{G}q$ | **Ind. Hyp.** |
| 2. | $\forall x \cdot (q \to \mathbf{G}q)$ | **GEN-$\forall$** 1 |
| 3. | $\forall x \cdot (q \to \mathbf{G}q) \to (\forall x \cdot q \to \forall x \cdot \mathbf{G}q)$ | **MON-$\forall$** |
| 4. | $\forall x \cdot q \to \forall x \cdot \mathbf{G}q$ | **R1-MP** 2, 3 |
| 5. | $\forall x \cdot \mathbf{G}q \to \mathbf{G}(\forall x \cdot q)$ | **BARC-G** |
| 6. | $\forall x \cdot q \to \mathbf{G}(\forall x \cdot q)$ | **HS** 4, 5 |

- $p \equiv q\mathbf{V}r$:

| | | |
|---|---|---|
| 1. | $q \to \mathbf{G}q$ | **Ind. Hyp.** |
| 2. | $\bot \to r$ | **REFL**, **NEG-L**, **R1-MP**, **D2-$\bot$** |
| 3. | $\mathbf{X}q \to q\mathbf{V}r$ | **R2-G** 2, **A4-GV**, **R1-MP**, **D7-X** |
| 4. | $\mathbf{GX}q \to \mathbf{G}(q\mathbf{V}r)$ | **R2-G** 3, **MON-G**, **R1-MP** |
| 5. | $\mathbf{GG}q \to \mathbf{GX}q$ | **RPL-GX**, **R2-G**, **MON-G**, **R1-MP** |
| 6. | $\mathbf{G}q \to \mathbf{GX}q$ | **IDEM-G**, **HS** 5 |
| 7. | $q \to \mathbf{G}(q\mathbf{V}r)$ | **HS** 1, 6; **HS** 4 |
| 8. | $\mathbf{G}(q \to \mathbf{G}(q\mathbf{V}r))$ | **R2-G** 7 |
| 9. | $\mathbf{X}q \to \mathbf{XG}(q\mathbf{V}r)$ | **MON-GX**, **R1-MP** 8 |
| 10. | $\mathbf{G}(\mathbf{G}(q\mathbf{V}r) \to q\mathbf{V}r)$ | **REFL-G**, **R2-G** |
| 11. | $\mathbf{XG}(q\mathbf{V}r) \to \mathbf{X}(q\mathbf{V}r)$ | **MON-GX**, **R1-MP** 10 |

| | | |
|---|---|---|
| 12. | $\mathbf{X}q \rightarrow \mathbf{X}(q\mathbf{V}r)$ | **HS** 9, 11 |
| 13. | $q\mathbf{V}r \rightarrow \mathbf{X}(q \vee r \wedge q\mathbf{V}r)$ | **FIX-V** |
| 14. | $\mathbf{X}(q \vee r \wedge q\mathbf{V}r) \rightarrow \mathbf{X}q \vee \mathbf{X}(r \wedge q\mathbf{V}r)$ | **A9-V**, **D7-X** |
| 15. | $\mathbf{X}r \wedge \mathbf{X}(q\mathbf{V}r) \rightarrow \mathbf{X}(q\mathbf{V}r)$ | **REFL**, **AND-L** |
| 16. | $\mathbf{X}(r \wedge q\mathbf{V}r) \rightarrow \mathbf{X}(q\mathbf{V}r)$ | **DIST-ANDX**, **R1-MP** 15 |
| 17. | $\mathbf{X}q \vee \mathbf{X}(r \wedge q\mathbf{V}r) \rightarrow \mathbf{X}(q\mathbf{V}r)$ | **OR-L** 11, 16 |
| 18. | $q\mathbf{V}r \rightarrow \mathbf{X}(q\mathbf{V}r)$ | **HS** 13, 14; **HS** 17 |
| 19. | $q\mathbf{V}r \rightarrow \mathbf{G}(q\mathbf{V}r)$ | **R1-G** 18, **A10-G**, **R1-MP** |

- $p \equiv \mathbf{A}q$:

| | | |
|---|---|---|
| 1. | $q \rightarrow \mathbf{G}q$ | **Ind. Hyp.** |
| 2. | $\mathbf{A}(q \rightarrow \mathbf{G}q)$ | **R4-A** 1 |
| 3. | $\mathbf{A}q \rightarrow \mathbf{AG}q$ | **A13-A**, **R1-MP** 2 |
| 4. | $\mathbf{AG}q \rightarrow \mathbf{GA}q$ | **COM-AG** |
| 5. | $\mathbf{A}q \rightarrow \mathbf{G}(\mathbf{A}q)$ | **HS** 3, 4 ■ **(RIGID)** |

Let us recall the main purpose of the proposition and theorems above. We wanted to establish a design discipline to support the verification of liveness properties. Now we can say it consists in the following steps: (i) if necessary, extend the given specification with the relation symbol and a suitable explicit definition; (ii) extend the specification with an arbitrarily chosen unconstrained flexible symbol via a faithful morphism; (iii) derive **IRR**, **APROG** and also **STAB** and **TERM** if required; (iv) derive the liveness property based on **WELL**. All these steps are justified by the previous results. A complete example will be provided in Chapter 3.

Some authors attempt to deal with the problem above in distinct ways. Lamport (1994) adopts a basic inference rule for well-founded induction but does not explain in full detail how the required premise in the rule is to be obtained. Abadi and Merz (1996) sketch a solution adopting the quantification over flexible variables of $TLA$, which is known to increase considerably the expressive power of any temporal proof calculus. On the other hand, it is not clear how this and the other logical connectives are related. Andréka *et al.* (1995) prefers to adopt a structural induction schema over so-called data-domains, which are specified in non-standard first-order temporal logic. All these authors have only considered the case in which a rigid relation symbol is given.

## 2.8 A Particular Model Theory

Semantic models for branching time such as transition systems and event structures abound in the literature. The following definition is of the first kind:

$$\begin{tikzcd}
 & & PBTL \arrow[dr] & \\
 & PLTL \arrow[ur] \arrow[dr] & & MSBTL \\
CPL \arrow[ur] \arrow[dr] & & MSLTL \arrow[ur] & \\
 & FOL \arrow[r] & MSFOL \arrow[u] &
\end{tikzcd}$$

Figure 2.14: Faithful logical system embeddings.

**Definition 2.8.1 (Branching Time Structure)** A *branching time structure* or *frame* is a tuple ($\alpha$, $\alpha_0$, $\rho$, $\Lambda$) where:

- $\alpha$ and $\alpha_0 \subseteq \alpha$ are sets of worlds and initial worlds respectively;
- $\rho : \alpha \to \mathcal{P}(\alpha)$ is the accessibility relation (a powerset function);
- $\Lambda$ is a non-empty set of possible behaviours. Each $L \in \Lambda$ is a function such that[12]: (i) $\mathsf{dom}\ L \subseteq \alpha$ and $\mathsf{cod}\ L \overset{\text{def}}{=} \mathbf{N}$; (ii) $L(w) = 0$ iff $w \in \alpha_0$; (iii) $\forall w, w' \in \mathsf{dom}\ L \cdot L(w) = L(w') \to w = w'$; (iv) $\forall n \in \mathsf{cod}\ L \cdot \exists w \in \mathsf{dom}\ L \cdot L(w) = n$; and (v) $\forall w, w' \in \mathsf{dom}\ L \cdot L(w') = L(w) + 1 \to w' \in \rho(w)$. $\square$

The sequences of worlds which determine behaviours in $\Lambda$ (not necessarily of any computer program) are in a one to one correspondence with the set of natural numbers, according to (iii) and (iv). Hence, each $L \in \Lambda$ is invertible and we shall use this fact to define the meaning of $\mathbf{A}$. It is also in this semantic way that problematic cyclic flows of time are avoided. Concerning the semantic assumptions over branching time structures proposed in the literature, it is easy to see that prefix (Stirling 1992), suffix and fusion (Emerson 1983) closures do not follow from our definition.

Based on branching time structures, signature symbols are interpreted as:

**Definition 2.8.2 (Interpretation Structure)** An *interpretation structure* for a signature $\Delta$ is a tuple $\theta = (T, U, G, A)$ where:

[12]Recall that we deal with $\omega$-long behaviours only, as explained in Section 2.4.

- $T$ is a branching time structure;
- $U$ maps each $s \in Sort(\Delta)$ to a non-empty collection $s_U$ and each $f \in Fun(\Delta)$ with $type(f) = \langle s_1, \ldots, s_n \rangle \rightarrow s$ to a function $f_U : s_{1_U} \times \ldots \times s_{n_U} \rightarrow s_U$;
- $G$ maps each $g \in Attr(\Delta)$ with $type(g) = \langle s_1, \ldots, s_n \rangle \rightarrow s$ to a function $G(g) : s_{1_U} \times \ldots \times s_{n_U} \rightarrow \alpha \rightarrow s_U$;
- $A$ maps each $a \in Act(\Delta)$ with $type(a) = \langle s_1, \ldots, s_n \rangle$ to a function $A(a) : s_{1_U} \times \ldots \times s_{n_U} \rightarrow \mathcal{P}(\alpha)$. $\square$

We adopt the interpretation structures above as models of logical formulas. As a result, whenever a formula has a model, the sets of worlds $\alpha$ and $\alpha_0$ in the underlying frame are not empty. Note how $\alpha$ appearing as an argument in the interpretation of some symbols is related to their flexible, time-dependent meaning. Interpreting symbols in $Act(\Delta)$ particularly shows that the respective actions may happen in parallel among themselves, in which case this is specified through the conjunction of their symbols, or with respect to other actions in the environment. This is much in keeping with the open but not necessarily interleaving semantics proposed in (Barringer 1987, Fiadeiro and Maibaum 1992). The notion of reduct of a model along a signature morphism will also be useful in our subsequent investigations:

**Definition 2.8.3 (Reduct of a model)** Given $\Delta_1 \xrightarrow{\tau} \Delta_2$ in morph $\mathbf{Sig}^{MSBTL}$ and an interpretation structure $\theta_2 = (T_2, U_2, G_2, A_2)$ for $\Delta_2$, the model $\theta_1 = (T_2, U_2 \circ \tau, G_2 \circ \tau, A_2 \circ \tau)$ is called the *$\tau$-reduct* of $\theta_2$. $\square$

We interpret terms as defined below. Because we have a first-order logic, it is first necessary to define how logical variables (which are not specified as part of signatures) are assigned to the elements of quantification domains. Assignments are alternatively called *valuations*:

**Definition 2.8.4 (Assignment)** Given an interpretation structure $\theta$ for a signature $\Delta$, an *assignment* $N$ for $\theta$ maps each set $Class(\Delta)_s$ to $s_U$. $\square$

**Definition 2.8.5 (Interpretation of Terms)** Given an interpretation structure $\theta = (T, U, G, A)$ for a signature $\Delta$ and an assignment $N$ for $\theta$, the function $[\![]\!]^{\theta,N} : \alpha \rightarrow s_U$ defined as follows is an *interpretation of terms* of sort $s \in Sort(\Delta)$ at a world $w \in \alpha$:

- $[\![x]\!]^{\theta,N}(w) \stackrel{\text{def}}{=} N(x)$ if $x \in Class(\Delta)_s$;

- $[\![f(t_1,\ldots,t_n)]\!]^{\theta,N}(w) \stackrel{\text{def}}{=} f_U([\![t_1]\!]^{\theta,N}(w),\ldots,[\![t_n]\!]^{\theta,N}(w))$;
- $[\![g(t_1,\ldots,t_n)]\!]^{\theta,N}(w) \stackrel{\text{def}}{=} (G(g)([\![t_1]\!]^{\theta,N}(w),\ldots,[\![t_n]\!]^{\theta,N}(w)))(w)$. □

We have said that the branching modality of our logic is to be interpreted with the help of an equivalence relation over behaviour prefixes. We define such a relation in a pointwise manner, in terms of the equivalence of worlds composing the possible behaviours of a structure, as follows:

**Definition 2.8.6 (Equivalent worlds)** Two worlds $\{w, w'\} \subset \alpha$ of a branching time structure $T = (\alpha, \alpha_0, \rho, \Lambda)$ in an interpretation $\theta = (T,\, U,\, A,\, G)$ for a signature $\Delta$ are said to be *equivalent*, $w \simeq w'$, if and only if

$$\forall g \in \mathit{Attr}(\Delta) \cdot \forall x_1,\ldots,x_n \cdot G(g)(x_1,\ldots,x_n)(w) = G(g)(x_1,\ldots,x_n)(w')$$
$$\forall a \in \mathit{Act}(\Delta) \cdot \forall x_1,\ldots,x_n \cdot w \in A(a)(x_1,\ldots,x_n) \Leftrightarrow w' \in A(a)(x_1,\ldots,x_n) \qquad \square$$

Indeed, $\simeq$ is an equivalence relation being reflexive, symmetric and transitive due to the equality and the biconditional in the sentences above. Since we choose the usual interpretation of logical formulas below, it is not difficult to conclude by induction that equivalent worlds satisfy the same set of state formulas, those formed out of variables, signature symbols and classical connectives only. Hence, two behaviours are considered to be equivalent up to a given moment if and only if they have identical past histories, i.e., they satisfy at each previous moment the same set of such formulas. This is again an equivalence relation because of the same property of $\simeq$. Note that these internal notions of equivalence are different from the usual external notion of *zig zags* (van Benthem 1984) because they relate states and behaviours of a model as opposed to the relations between models, interpretation structures, defined by zig zags.

We use the above to define the satisfaction of logical formulas:

**Definition 2.8.7 (Satisfaction of Formulas)** Given a signature $\Delta$, the *satisfaction* of a $\Delta$-formula at world $w_i$ of a behaviour $L$ (i.e., $w_i \in \mathsf{dom}\ L$) by a structure $\theta = (T,\, U,\, G,\, A)$ with assignment $N$ is defined as follows:

**S1.** $(\theta, N, L, w_i) \models a(t_1,\ldots,t_n)$ iff $w_i \in A(a)([\![t_1]\!]^{\theta,N}(w_i),\ldots,[\![t_n]\!]^{\theta,N}(w_i))$;

**S2.** $(\theta, N, L, w_i) \models \neg p$ iff it is not the case that $(\theta, N, L, w_i) \models p$;

**S3.** $(\theta, N, L, w_i) \models p \rightarrow q$ iff $(\theta, N, L, w_i) \models p$ implies $(\theta, N, L, w_i) \models q$;

**S4.** $(\theta, N, L, w_i) \models \forall x \cdot p$ iff for every $v \in \mathsf{cod}\ N$ and assignment $N_v$ for $\theta$ such that $N_v(y) = N(y)$ if $y \neq x$ and $N_v(y) = v$ otherwise, $(\theta, N_v, L, w_i) \models p$;

**S5.** $(\theta, N, L, w_i) \models (t_1 = t_2)$ iff $[\![t_1]\!]^{\theta,N}(w_i) = [\![t_2]\!]^{\theta,N}(w_i)$;

**S6.** $(\theta, N, L, w_i) \models \mathbf{beg}$ iff $L(w_i) = 0$;

**S7.** $(\theta, N, L, w_i) \models p\mathbf{V}q$ iff there is $w_j \in \mathsf{dom}\ L$ with $L(w_i) < L(w_j)$, $(\theta, N, L, w_j) \models p$ and $(\theta, N, L, w_k) \models q$ for any $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(w_k) < L(w_j)$;

**S8.** $(\theta, N, L, w_i) \models \mathbf{A}p$ iff for every $L_j \in \Lambda$ such that $w_k \simeq (L_j^{-1} \circ L)(w_k)$ for each $w_k \in \mathsf{dom}\ L$ with $L(w_k) < L(w_i)$, $(\theta, N, L_j, (L_j^{-1} \circ L)(w_i)) \models p$. □

The definition of satisfiability above determines a floating interpretation for our logic, according to the terminology of Manna and Pnueli (1989). That is, the initial instant has no special significance in the interpretation, even though it is represented as the logical connective **beg**. Based on this definition, it is not difficult to prove by structural induction that:

**Proposition 2.8.8 (Equivalent worlds satisfy the same state formulas)** Given an interpretation structure $\theta = ((\alpha, \alpha_0, \rho, \Lambda), U, G, A)$ for a signature $\Delta$ and an assignment $N$ for $\theta$, $w \simeq w'$ iff for any state formula $p$, $(\theta, N, L, w) \models p$ iff $(\theta, N, L', w') \models p$ for any $\{L, L'\} \subseteq \Lambda$ such that $w \in \mathsf{dom}\ L$, $w' \in \mathsf{dom}\ L'$. ■

We define an ascending series of degrees of validity as suggested by Chellas (1980). Definition 2.8.7 corresponds to *satisfiability.* We say that a $\Delta$-formula $p$ is *locally true* in an interpretation structure $\theta = (T, U, G, A)$ for $\Delta$ at world $w$ of a behaviour $L$ if and only if for every assignment $N$, $(\theta, N, L, w) \models p$. A sentence $p$, a formula such that $Free(p) = \{\}$, is *true* in $\theta$ if and only if locally true in each behaviour $L$ and world $w$ such that $w \in \mathsf{dom}\ L$. We write $\theta \models p$ in this case. A semantic consequence relation over a model $\theta$, $\Psi \models^{\theta}_{\Delta} p$, is simply defined by saying that $\theta \models q$ for every sentence $q \in \Psi$ implies $\theta \models p$. If we require this for every model, we obtain the semantic consequence relation $\Psi \models_{\Delta} p$. Completing our hierarchy, $p$ is said to be *valid* in $T$ if and only if true in any $\theta$ based on $T$. A sentence is considered to be *universally valid* if and only if it is valid in any branching time structure.

As a last word in this section, it is important to mention that restricting the language and the interpretation structures above in some particular ways result in models of other logics studied in this chapter. For instance, if we forget assignments and quantifiers we obtain propositional branching time logic models. Forgetting the branching modality and that interpretations consist of non-empty collections of behaviours, we obtain models of linear time logic by picking up single elements from each such collection of behaviours. Models for classic propositional logic are obtained by forgetting completely the time dimension and the respective connectives.

## 2.9 Some General Logical Results

Our main purpose in this section is to show that $MSBTL$ is a logical system in the precise sense of Definition 2.2.7. We have already shown that all the required properties to define a full entailment system are fulfilled. Now we have to ensure that $MSBTL$ determines an institution.

We first deal with the problem of defining a category of models associated to each signature. The structure of the objects in this category has already been defined in Section 2.8 in the form of interpretation structures. The collection of functions admitted as morphisms in such categories normally results from an arbitrary decision concerning the particular modal logic, so we adopt here an extended version of the so-called p-morphisms (Segeberg 1970). Given a signature $\Delta$ with two interpretation structures $\theta_i = (T_i, U_i, G_i, A_i)$ such that $T_i = (\alpha_i, \alpha_{0_i}, \rho_i, \Lambda_i)$, $1 \leq i \leq 2$, a first-order p-morphism (fp-morphism for short) $\tau : \theta_1 \to \theta_2$ is a pair $(\tau_U, \tau_\alpha)$, where $\tau_\alpha : \alpha_1 \to \alpha_2$, $\tau_U : U_1 \to U_2$ and $\tau_U$ is a model homomorphism of the classical first-order functional calculus (recall that predicates have a temporalised interpretation here). It is important to mention that the following condition is required from any such an homomorphism. For every $f \in Funct(\Delta)$, $s_i \in Sort(\Delta)$, the following diagram commutes:

$$\begin{tikzcd} s^*_{U_1} \arrow[r, "\tau^*_U"] \arrow[d, "f_U"'] & s^*_{U_2} \arrow[d, "\tau_U(f_U)"] \\ s_{U_1} \arrow[r, "\tau_U"'] & s_{U_2} \end{tikzcd} \tag{2.9.1}$$

Moreover, for each fp-morphism $(\tau_U, \tau_\alpha)$, $\tau_\alpha$ is required to map behaviours onto behaviours and the following conditions must be obeyed, for every world $\{w_i, w'_i\} \subseteq \alpha_i$, $1 \leq i \leq 2$:

(i) $w_1 \in \alpha_{0_1} \Rightarrow \tau_\alpha(w_1) \in \alpha_{0_2}$;

(ii) $w'_1 \in \rho_1(w_1) \Rightarrow \tau_\alpha(w'_1) \in \rho_2(\tau_\alpha(s_1))$;

(iii) $w'_2 \in \rho_2(\tau_\alpha(w_1)) \Rightarrow \exists w'_1 \cdot w'_1 \in \rho_1(w_1) \wedge \tau_\alpha(w'_1) = w'_2$;

(vi) For every $g \in Attr(\Delta)$ such that $arity(g) = n$,

$$\forall x_1, \ldots, x_n \cdot \tau_U(G_1(g)(x_1, \ldots, x_n)(w_1)) = G_2(g)(\tau_U(x_1), \ldots, \tau_U(x_n))(\tau_\alpha(w_1));$$

(v) For every $a \in Act(\Delta)$ such that $arity(a) = n$,

$$\forall x_1, \ldots, x_n \cdot w_1 \in A_1(a)(x_1, \ldots, x_n) \Leftrightarrow \tau_\alpha(w_1) \in A_2(a)(\tau_U(x_1), \ldots, \tau_U(x_n)).$$

With the definition above, it is not difficult to check that **ASS** and **ID** are obtained, validating the following proposition:

**Proposition 2.9.1 (Categories of $MSBTL$ Models)** The collections of interpretation structures and fp-morphisms for each signature $\Delta$ in obj $\mathbf{Sig}^{MSBTL}$ determine a $\Delta$-indexed family of categories of models $\mathbf{Mod}_\Delta^{MSBTL}$. ■

Note in the definition of institution that each signature $\Delta$ in obj $\mathbf{Sig}^{MSBTL}$ is assigned to a category of models $\mathbf{Mod}_\Delta^{MSBTL}$ by a contravariant functor $Mod : \mathbf{Sig}^{MSBTL} \to \mathbf{Cat}^{op}$. Considering the fp-morphisms defined above, this functor can be defined as follows:

1. $Mod(\Delta) \overset{\text{def}}{=} \mathbf{Mod}_\Delta^{MSBTL}$ for each $\Delta$ in obj $\mathbf{Sig}^{MSBTL}$;
2. $Mod(\sigma : \Delta_1 \to \Delta_2) \overset{\text{def}}{=} Mod(\sigma) : Mod(\Delta_2) \to Mod(\Delta_1)$ for each $\Delta_i$ in obj $\mathbf{Sig}^{MSBTL}$, $1 \leq i \leq 2$, and each $\mathbf{Sig}^{MSBTL}$-morphism $\sigma$;
3. For each $\Delta_1 \xrightarrow{\sigma} \Delta_2$ in morph $\mathbf{Sig}^{MSBTL}$, the following diagram commutes for each $\theta_i = (T_i, U_i, G_i, A_i)$ in obj $\mathbf{Mod}_{\Delta_i}^{MSBTL}$, $1 \leq i \leq 2$, and each pair $(X, Y)$ in the set $\{(Sort, U), (Funct, U), (Attr, G), (Act, A)\}$:

$$\begin{tikzcd}
X_1(\Delta_1) \arrow[r, "\sigma"] \arrow[d, "Y_1"'] & (\sigma \circ X_1)(\Delta_1) \arrow[d, "Y_2"] \\
(Y_1 \circ X_1)(\Delta_1) & (Y_2 \circ \sigma \circ X_1)(\Delta_1) \arrow[l, "Mod(\sigma)"]
\end{tikzcd} \tag{2.9.2}$$

4. For each $\Delta_1 \xrightarrow{\sigma} \Delta_2$ in morph $\mathbf{Sig}^{MSBTL}$, $\theta_i = (T_i, U_i, G_i, A_i)$ in obj $\mathbf{Mod}_{\Delta_i}^{MSBTL}$, $T_i = (\alpha_i, \alpha_{0_i}, \rho_i, \Lambda_i)$, $1 \leq i \leq 2$, and each $\theta_1 \xrightarrow{\tau} \theta_2$ in morph $\mathbf{Mod}_{\Delta_i}^{MSBTL}$, the following diagram commutes for each pair $(X, \varphi)$ in the set $\{(U, \tau_U), (\alpha, \tau_\alpha)\}$:

$$\begin{tikzcd}
X_1 \arrow[r, "\varphi"] \arrow[d, "Mod(\sigma)"'] & X_2 \arrow[d, "Mod(\sigma)"] \\
Mod(\sigma)(X_1) \arrow[r, "Mod(\sigma)(\varphi)"'] & Mod(\sigma)(X_2)
\end{tikzcd} \tag{2.9.3}$$

The last two conditions above are to guarantee that the structure of each category of models is preserved by the model functor.

**Lemma 2.9.2 (*MSBTL* Institution)** With the definitions provided above, the tuple ($\mathbf{Sig}^{MSBTL}$, $\mathcal{G}^{MSBTL}$, $Mod$, $\models^{MSBTL}$) is an institution.

Proof: We only need to verify that the satisfaction condition is fulfilled. That is, for every $\Delta_1 \xrightarrow{\sigma} \Delta_2$ in morph $\mathbf{Sig}^{MSBTL}$, $p \in \mathcal{G}^{MSBTL}(\Delta_1)$ and $\theta$ in obj $\mathbf{Mod}_{\Delta_2}$,

$$\theta \models^{MSBTL}_{\Delta_2} \sigma^{\#}(p) \Leftrightarrow Mod(\sigma)(\theta) \models^{MSBTL}_{\Delta_1} p$$

In particular, $p$ can only be a sentence in this assertion since the notion of truth in a model as defined in Section 2.8 makes this requirement.

Without loss of generality, we can sketch this proof considering that $\Delta_1$ and $\Delta_2$ are the same signature. This is due to the compositional definition of term, sentence and assignment functors and also due to (2.9.2) and (2.9.3), which guarantee that each category of models has an exact image along signature morphisms. That is, the internal structure of each model is matched exactly (2.9.2) and the same happens to the internal structure of each fp-morphism (2.9.3). Note, however, that more objects and morphisms may be present in the source category of models and more symbols may exist in the target signature. These do not create a problem because we only need to work with the image of $\sigma$ and the respective reducts proving the satisfaction condition. The remainder of the co-domain of these morphisms can be safely ignored.

Now we can develop the rest of the proof by a structural induction argument on the grammar of the language. Consider a fixed signature $\Delta$ with models $\theta_i = (T_i, U_i, G_i, A_i)$ such that $T_i = (\alpha_i, \alpha_{0_i}, \rho_i, \Lambda_i)$, $1 \leq i \leq 2$, and $\theta_2 \xrightarrow{\tau} \theta_1$. We wish to show that for any $L_2 \in \Lambda_2$ and any $w_2 \in$ dom $L_2$, $(\theta_2, N', L_2, w_2) \models p$ for any assignment $N'$ for $\theta_2$ if and only if $(\theta_1, N, \tau^{\#}(L_2), \tau(w_2)) \models p$ for any assignment $N$ for $\theta_1$. This is an extension of the well-known p-morphism lemma in the modal logics literature (Goldblatt 1992). We examine in the subsequent paragraph the base case of the induction argument and then proceed with some interesting cases of the induction step.

Given rigid terms $t_i \in Term(\Delta)_s$, $1 \leq i \leq 2$, $s \in Sort(\Delta)$, we have $(\theta_2, N', L_2, w_2) \models (t_1 = t_2)$ if and only if $[\![t_1]\!]^{\theta_2,N'}(w_2) = [\![t_2]\!]^{\theta_2,N'}(w_2)$. Because each $t_i$ is assumed to be rigid, their interpretations do not depend on the underlying world. Assume in addition that each $t_i$ is a constant, i.e., $t_i \in Funct(\Delta)$ such that $type(t_i) = \epsilon \rightarrow s$, and then their interpretations will not depend on the assignment as well. Due to the functionality of $\tau_U$, $[\![t_1]\!]^{\theta_1,N}(\tau(w_2)) = [\![t_2]\!]^{\theta_1,N}(\tau(w_2))$. Hence, $(\theta_1, N, \tau^{\#}(L_2), \tau(w_2)) \models (t_1 = t_2)$. The converse is proved observing that for constants the homomorphism condition (2.9.1) requires that $[\![t_1]\!]^{\theta_2,N'}(w_2) = \tau([\![t_1]\!]^{\theta_2,N'}(w_2))$. For non-rigid constants, case (iv) in the definition of fp-morphism guarantees that the biconditional above can be obtained. This rationale easily generalises to any kind of term and to state formulas as well.

Let us examine the temporal formulas. Assume that $(\theta_2, N', L_2, w_2) \models$ **beg**. Linking **S6**, requirements (ii) in Definition 2.8.1 and (i) in the definition of fp-morphism, we infer that $\tau(w_2) \in \alpha_{0_1}$. Then, the first two applied in the inverse order also justify $(\theta_1, N, \tau^{\#}(L_2), \tau(w_2)) \models$ **beg**. For the converse, suppose that $(\theta_1, N, \tau^{\#}(L_2), \tau(w_2)) \models$ **beg** but $(\theta_2, N', L_2, w_2) \models$ **beg** is not the case. The first conjunct ensures that $\tau(w_2)$ is the first element in $\mathsf{dom}\ \tau^{\#}(L_2)$. Moreover, the second one shows that $\exists w_3 \in \mathsf{dom}\ L_2 \cdot L_2(w_2) = L(w_3) + 1$. Using condition (ii) in the definition of fp-morphisms shows that $\tau(w_2)$ is not the first element in $\mathsf{dom}\ \tau^{\#}(L_2)$, which generates a contradiction. We conclude that $(\theta_2, N', L_2, w_2) \models$ **beg** iff $(\theta_1, N, \tau^{\#}(L_2), \tau(w_2)) \models$ **beg**. The case of $p\mathbf{V}q$ is developed based on the back and forth conditions (iii) and (ii). The $\mathbf{A}p$ case is developed based on the fact that each $\tau_\alpha$ is onto concerning behaviours.

We have concluded that $(\theta_2, N', L_2, w_2) \models p$ iff $(\theta_1, N, \tau^{\#}(L_2), \tau(w_2)) \models p$ for any $p \in \mathcal{G}(\Delta)$. Extending this partial result to the case where we have models for different signatures, assume in addition that $\Delta_1 \xrightarrow{\sigma} \Delta_2$, $\theta_1$ in $\mathsf{obj}\ \mathbf{Mod}_{\Delta_2}$, $Mod(\sigma) = \tau$ and $\tau(\theta_2) = \theta_1$. Applying the different morphisms and functors involved in this situation, we obtain $\theta_1 \models_{\Delta_2} \sigma^{\#}(p)$ if and only if $\theta_2 \models_{\Delta_1} p$. Therefore, the tuple above is an institution. ■ ($MSBTL$ **Institution**)

We turn to the verification of the soundness condition in Definition 2.2.5:

**Lemma 2.9.3 (Soundness of $MSBTL$)** $MSBTL$ is sound.

Proof: We show in the usual way, based on the notion of satisfaction, that each logical axiom is universally valid and the application of each inference rule preserves validity, meaning that valid premises imply valid conclusions. We present here the interesting cases only, leaving the verification of the remaining cases for Appendix II. An additional structural induction argument on our Hilbert-style proofs will suffice to guarantee that each entailment preserves validity.

We prove that each inference rule in Figure 2.15 preserves validity as follows, assuming that an underlying signature $\Delta$ with a branching time structure $T$ are given and also that the notion of satisfaction for the derived connectives has already been worked out. Whenever necessary, we denote by $\theta'_T$ an interpretation structure which is obtained from $\theta = (T, U, G, A)$ by varying all the components apart from the frame $T$.

**(R1-MP)** Assume that (i) $(\theta_T, N, L, w_i) \models p$ for any $\theta_T$, $N$, $L$, $w_i \in \mathsf{dom}\ L$ and (ii) $(\theta'_T, N', L', w'_i) \models p \rightarrow q$ for any $\theta_T$, $N$, $L$ and $w_i \in \mathsf{dom}\ L$. From

*Syntax:* For a given signature $\Delta$, we have:

| | |
|---|---|
| (**Terms**) | $T ::= x$ (variables) $\mid f(T_1, \ldots, T_m)$ (functions) $\mid g(T_1, \ldots, T_n)$ (attributes) |
| (**Atoms**) | $A ::= T_s = T_s$ (equality) $\mid a(T_1, \ldots, T_n)$ (actions) |
| (**Formulas**) | $F ::= A \mid \neg F \mid F \to F \mid (F) \mid \forall x \cdot F \mid \mathbf{beg} \mid (F)\mathbf{V}(F) \mid \mathbf{A}(F)$ |

*Definitions:*

| | | | |
|---|---|---|---|
| $(\mathbf{D1} - \top)$ | $\top \overset{\text{def}}{=} p \to p$ | $(\mathbf{D2} - \bot)$ | $\bot \overset{\text{def}}{=} \neg\top$ |
| $(\mathbf{D3} - \mathbf{OR})$ | $(p \vee q) \overset{\text{def}}{=} (\neg p \to q)$ | $(\mathbf{D4} - \mathbf{AND})$ | $(p \wedge q) \overset{\text{def}}{=} \neg(p \to \neg q)$ |
| $(\mathbf{D5} - \mathbf{IFF})$ | $(p \leftrightarrow q) \overset{\text{def}}{=} (p \to q) \wedge (q \to p)$ | $(\mathbf{D6} - \mathbf{X})$ | $\mathbf{X}p \overset{\text{def}}{=} p\mathbf{V}\bot$ |
| $(\mathbf{D7} - \mathbf{U})$ | $p\mathbf{U}q \overset{\text{def}}{=} q \vee (p \wedge q\mathbf{V}p)$ | $(\mathbf{D8} - \mathbf{F})$ | $\mathbf{F}p \overset{\text{def}}{=} \top\mathbf{U}p$ |
| $(\mathbf{D9} - \mathbf{G})$ | $\mathbf{G}p \overset{\text{def}}{=} \neg\mathbf{F}(\neg p)$ | $(\mathbf{D10} - \mathbf{W})$ | $p\mathbf{W}q \overset{\text{def}}{=} \mathbf{G}p \vee p\mathbf{U}q$ |
| $(\mathbf{D11} - \mathbf{E})$ | $\mathbf{E}p \overset{\text{def}}{=} \neg\mathbf{A}(\neg p)$ | $(\mathbf{D12} - \exists)$ | $\exists x \cdot p \overset{\text{def}}{=} \neg\forall x \cdot \neg p$ |
| $(\mathbf{D13} - \mathbf{NEQ})$ | $t_1 \neq t_2 \overset{\text{def}}{=} \neg(t_1 = t_2)$ | | |
| $(\mathbf{D14} - \mathbf{UNI})$ | $\exists!\, x \cdot p[x] \overset{\text{def}}{=} \exists x \cdot (p[x] \wedge \forall y \cdot p[y] \to x = y)$ | | |

*Axioms:* In **A20** and **A23**, $x \notin \mathit{Free}(p)$; in **A19**, **A24-5** and **A27**, $(\mathcal{E}(t) \cup \mathcal{E}(t_1) \cup \mathcal{E}(t_2)) \cap \mathit{Attr}(\Delta) = \{\,\}$; $p \in \mathit{Atom}(\Delta)$ in **A22**:

| | |
|---|---|
| $(\mathbf{A1} - \mathbf{I})$ | $p \to (q \to p)$ |
| $(\mathbf{A2} - \mathbf{I})$ | $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$ |
| $(\mathbf{A3} - \mathbf{N})$ | $(\neg p \to \neg q) \to (q \to p)$ |
| $(\mathbf{A4} - \mathbf{GV})$ | $\mathbf{G}(p \to q) \to (p\mathbf{V}r \to q\mathbf{V}r)$ |
| $(\mathbf{A5} - \mathbf{GV})$ | $\mathbf{G}(p \to q) \to (r\mathbf{V}p \to r\mathbf{V}q)$ |
| $(\mathbf{A6} - \mathbf{V})$ | $p\mathbf{V}q \to p\mathbf{V}(q \wedge p\mathbf{V}q)$ |
| $(\mathbf{A7} - \mathbf{V})$ | $(p \wedge q\mathbf{V}p)\mathbf{V}p \to q\mathbf{V}p$ |
| $(\mathbf{A8} - \mathbf{V})$ | $p\mathbf{V}q \wedge r\mathbf{V}s \to (p \wedge r)\mathbf{V}(q \wedge s) \vee (p \wedge s)\mathbf{V}(q \wedge s) \vee (q \wedge r)\mathbf{V}(q \wedge s)$ |
| $(\mathbf{A9} - \mathbf{V})$ | $(p \vee q)\mathbf{V}r \to p\mathbf{V}r \vee q\mathbf{V}r$ |
| $(\mathbf{A10} - \mathbf{G})$ | $\mathbf{G}(p \to \mathbf{X}p) \to (p \to \mathbf{G}p)$ |
| $(\mathbf{A11} - \mathbf{X})$ | $\mathbf{X}\top$ |
| $(\mathbf{A12} - \mathbf{Xbeg})$ | $\neg\mathbf{X}(\mathbf{beg})$ |
| $(\mathbf{A13} - \mathbf{A})$ | $\mathbf{A}(p \to q) \to (\mathbf{A}p \to \mathbf{A}q)$ |
| $(\mathbf{A14} - \mathbf{A})$ | $\mathbf{A}p \to p$ |
| $(\mathbf{A15} - \mathbf{EA})$ | $\mathbf{E}p \to \mathbf{AE}p$ |
| $(\mathbf{A16} - \mathbf{EV})$ | $(\mathbf{E}p)\mathbf{V}q \to \mathbf{E}(p\mathbf{V}q)$ |
| $(\mathbf{A17} - \mathbf{AU})$ | $\mathbf{A}(p \to \mathbf{X}(q\mathbf{U}p)) \to (p \to \mathbf{XA}(q\mathbf{U}p))$ |
| $(\mathbf{A18} - \mathbf{Ebeg})$ | $\mathbf{E}(\mathbf{beg}) \to \mathbf{beg}$ |
| $(\mathbf{A19} - \forall)$ | $(\forall x \cdot p[x]) \to p[x \backslash t]$ |
| $(\mathbf{A20} - \forall)$ | $\forall x \cdot (p \to q) \to (p \to \forall x \cdot q)$ |
| $(\mathbf{A21} - \mathbf{EQ})$ | $t = t$ |
| $(\mathbf{A22} - \mathbf{EQ})$ | $t_1 = t_2 \to (p\{q \backslash t_1\} \to p\{q \backslash t_2\})$ |
| $(\mathbf{A23} - \exists\mathbf{V})$ | $(\exists x \cdot q)\mathbf{V}p \to \exists x \cdot q\mathbf{V}p$ |
| $(\mathbf{A24} - \mathbf{EQG})$ | $t_1 = t_2 \to \mathbf{G}(t_1 = t_2)$ |
| $(\mathbf{A25} - \mathbf{NEQG})$ | $t_1 \neq t_2 \to \mathbf{G}(t_1 \neq t_2)$ |
| $(\mathbf{A26} - \forall\mathbf{A})$ | $\forall x \cdot \mathbf{A}p \to \mathbf{A}(\forall x \cdot p)$ |
| $(\mathbf{A27} - \mathbf{EQA})$ | $t_1 = t_2 \to \mathbf{A}(t_1 = t_2)$ |

*Inference Rules:* In **R5**, we consider that $x \notin \mathit{Free}(p)$:

| | | | |
|---|---|---|---|
| $(\mathbf{R1} - \mathbf{MP})$ | $\{p, p \to q\} \Vdash q$ | $(\mathbf{R2} - \mathbf{G})$ | $\{p\} \Vdash \mathbf{G}p$ |
| $(\mathbf{R3} - \mathbf{begG})$ | $\{\mathbf{beg} \to \mathbf{G}p\} \Vdash p$ | $(\mathbf{R4} - \mathbf{A})$ | $\{p\} \Vdash \mathbf{A}p$ |
| $(\mathbf{R5} - \forall)$ | $\{p \to q\} \Vdash p \to \forall x \cdot q$ | | |

Figure 2.15: Definition of $MSBTL$.

(ii) and **S3**, we infer that $(\theta_T, N, L, w_i) \models p$ implies $(\theta'_T, N', L', w'_i) \models q$, moving in this way the quantification over interpretations, assignments, behaviours and worlds to range over each instance of the satisfaction relation in isolation. Using (i), we conclude that $(\theta'_T, N', L', w'_i) \models q$ for any $\theta'_T$, $N'$, $L'$ and $w'_i \in \mathsf{dom}\ L'$;

**(R2-G)** Assume that $(\theta_T, N, L, w_i) \models p$ for any $\theta_T$, $N$, $L$ and $w_i \in \mathsf{dom}\ L$. For a fixed $L$, we consequently have for every $w_j \in \mathsf{dom}\ L$ that $(\theta_T, N, L, w_j) \models p$, which is equivalent to saying that $(\theta_T, N, L, w_i) \models \mathbf{G}p$ for any $\theta_T$, $N$, $L$ and $w_i \in \mathsf{dom}\ L$, according to the definition of satisfaction of $\mathbf{G}p$;

**(R3-begG)** Assume that $(\theta_T, N, L, w_i) \models \mathbf{beg} \rightarrow \mathbf{G}p$ for any $\theta_T$, $N$, $L$ and $w_i \in \mathsf{dom}\ L$. In particular, for $w_0 \in \mathsf{dom}\ L$ such that $L(w_0) = 0$, we have $(\theta_T, N, L, w_0) \models \mathbf{beg} \rightarrow \mathbf{G}p$. An application of **S3** shows that $(\theta_T$, $N$, $L$, $w_0) \models \mathbf{beg}$ implies $(\theta_T, N, L, w_0) \models \mathbf{G}p$, but the antecedent of this conditional is evident given **S6** and the definition of $w_0$. From the consequent of the conditional and the definition of satisfaction of $\mathbf{G}p$, we conclude that $(\theta_T, N, L, w_i) \models p$ for any $\theta_T$, $N$, $L$ and $w_i \in \mathsf{dom}\ L$;

**(R4-A)** Assume that $(\theta_T, N, L, w_i) \models p$ for any $\theta_T$, $N$, $L$ and $w_i \in \mathsf{dom}\ L$. For a fixed $L$, we have, for every $L_i \in \Lambda$ such that $s_k \simeq (L_i^{-1} \circ L)(w_k)$ for each $w_k \in \mathsf{dom}\ L$ with $L(w_k) < L(w_i)$, that $(\theta_T, N, L_i, (L_i^{-1} \circ L)(w_i)) \models p$, based on our assumption and that each $L_i$ is invertible. From **S8**, we conclude that $(\theta_T, N, L, w_i) \models \mathbf{A}p$ for any $\theta_T$, $N$, $L$ and $w_i \in \mathsf{dom}\ L$;

**(R5-$\forall$)** Assume that $(\theta_T, N, L, w_i) \models p \rightarrow q$ for any $\theta_T$, $N$, $L$ and $w_i \in \mathsf{dom}\ L$ and also that $x \notin \mathit{Free}(p)$. So, from **S3**, (i) $(\theta_T, N, L, w_i) \models p$ implies $(\theta_T, N, L, w_i) \models q$. Now, for each $v \in \mathsf{cod}\ N$, define $N_v$ as $N_v(y) \stackrel{\text{def}}{=} N(y)$ if $y \neq x$ or $N_v(y) \stackrel{\text{def}}{=} v$ otherwise. Note that, because $x \notin \mathit{Free}(p)$, (ii) $(\theta_T, N, L, w_i) \models p$ implies $(\theta_T, N_v, L, w_i) \models p$, by structural induction on the notions of interpretation and satisfaction based on the definitions of $\mathit{Free}$ and assignments. Substituting $N$ for $N_v$ in (i), we obtain (iii) $(\theta_T, N_v, L, w_i) \models p$ implies $(\theta_T, N_v, L, w_i) \models q$. Moving the quantification over $N_v$ inwards in (ii) and connecting this statement to (iii), we infer that $(\theta_T, N, L, w_i) \models p$ implies $(\theta_T, N_v, L, w_i) \models q$ for any $N_v$ defined as above, which means that $(\theta_T, N, L, w_i) \models p \rightarrow \forall x \cdot q(x)$ for any $\theta_T$, $N$, $L$ and $w_i \in \mathsf{dom}\ L$ by applying **S3** and **S4**.

The universal validity of each logical axiom listed in Figure 2.15 is verified as follows, assuming generic $\theta$, $N$, $L$ and $s_i \in \mathsf{dom}\ L$ for $\Delta$:

**(A1-I)** Suppose that (i) $(\theta, N, L, w_i) \models p$ and (ii) it is not the case that $(\theta, N, L, w_i) \models q \to p$. From (ii) and **S3**, we infer that it is not true that $(\theta, N, L, w_i) \models q$ implies $(\theta, N, L, w_i) \models p$. So, we have $(\theta, N, L, w_i) \models q$ but $(\theta, N, L, w_i) \models p$ does not hold, which contradicts (i). Therefore, $(\theta, N, L, w_i) \models p$ implies $(\theta, N, L, w_i) \models q \to p$ and we conclude that $(\theta, N, L, w_i) \models p \to (q \to p)$ using **S3**;

**(A4-GV)** Suppose that (i) $(\theta, N, L, w_i) \models \mathbf{G}(p \to q)$ and (ii) $(\theta, N, L, w_i) \models p\mathbf{V}r \to q\mathbf{V}r$ does not hold. From (ii) and **S3**, we have $(\theta, N, L, w_i) \models p\mathbf{V}r$ but $(\theta, N, L, w_i) \models q\mathbf{V}r$ is not the case. According to **S7**, this means that (iii) there is an $w_j \in \mathsf{dom}\ L$ with $L(w_i) < L(w_j)$ such that $(\theta, N, L, w_j) \models p$ and $(\theta, N, L, w_k) \models r$ for any $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(w_k) < L(w_j)$, and (iv) for every $w_m \in \mathsf{dom}\ L$ with $L(w_i) < L(w_m)$, $(\theta, N, L, w_m) \models q$ and $(\theta, N, L, w_n) \models r$ for any $w_n \in \mathsf{dom}\ L$ where $L(w_i) < L(w_n) < L(w_m)$ are not both true. In addition, the definition of satisfaction of $\mathbf{G}p$, (i) and **S3** leads to (v) $(\theta, N, L, w_j) \models p$ implies $(\theta, N, L, w_j) \models q$ for any $w_j \in \mathsf{dom}\ L$ such that $L(w_i) \leq L(w_j)$. Applying the first half of (iii) in (v), we infer that $(\theta, N, L, w_j) \models q$. For $w_m = w_j$, when we conjoin this partial result to (iv), we obtain a contradiction. We conclude, from the negation of our assumption and **S3**, that $(\theta, N, L, w_i) \models \mathbf{G}(p \to q) \to (p\mathbf{V}r \to q\mathbf{V}r)$;

**(A6-V)** Suppose that (i) $(\theta, N, L, w_i) \models p\mathbf{V}q$. From (i) and **S7**, we infer that (ii) there is $w_j \in \mathsf{dom}\ L$ with $L(w_i) < L(w_j)$ such that $(\theta, N, L, w_j) \models p$ and $(\theta, N, L, w_k) \models q$ for any $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(w_k) < L(w_j)$. Hence, for each $w_m$ such that $L(w_k) < L(w_m) < L(w_j)$, we know from (ii) that there is an $w_j \in \mathsf{dom}\ L$ with $L(w_m) < L(w_j)$ such that $(\theta, N, L, w_j) \models p$ and $(\theta, N, L, w_n) \models q$ for any $w_n \in \mathsf{dom}\ L$ where $L(w_m) < L(w_n) < L(w_j)$. We conclude, using the definition of satisfaction of $\wedge$ together with **S7** and **S3**, that $(\theta, N, L, w_i) \models p\mathbf{V}q \to p\mathbf{V}(q \wedge p\mathbf{V}q)$;

**(A8-V)** Suppose that (i) $(\theta, N, L, w_i) \models p\mathbf{V}q \wedge r\mathbf{V}s$. From (i), the definition of satisfaction of $\wedge$ and **S7**, we infer that (ii) there is $w_j \in \mathsf{dom}\ L$ with $L(w_i) < L(w_j)$ such that $(\theta, N, L, w_j) \models p$ and $(\theta, N, L, w_k) \models q$ for any $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(w_k) < L(w_j)$, and (iii) there is $w_l \in \mathsf{dom}\ L$ with $L(w_i) < L(w_l)$ such that $(\theta, N, L, w_l) \models r$ and $(\theta, N, L, w_m) \models s$ for any $w_m \in \mathsf{dom}\ L$ where $L(w_i) < L(w_m) < L(w_l)$. Let $w_n = \mathsf{min}\ (w_j, w_l)$. It is easy to see from the second half of (ii) and (iii) that $(\theta, N, L, w_o) \models q$ and $(\theta, N, L, w_o) \models r$ for any $w_o \in \mathsf{dom}\ L$ where $L(w_i) < L(w_o) < L(w_n)$. Now, if $w_j = w_l$, from the first half of (ii) and (iii), there is an $w_n$ such that $(\theta, N, L, w_n) \models p$ and $(\theta, N, L, w_n) \models r$. Alternatively, if $w_j < w_l$, from the first half of (ii) and the second half (iii), there is an $w_n$ such that $(\theta, N, L, w_n) \models p$ and $(\theta, N, L, w_n) \models s$. Otherwise, $(\theta, N, L, w_n) \models q$ and $(\theta, N, L, w_n) \models s$. Many applications of the definition of satisfaction of $\wedge$ and $\vee$ together with **S7** allow us to conclude that $(\theta, N, L, w_i) \models p\mathbf{V}q \wedge r\mathbf{V}s \rightarrow (p \wedge r)\mathbf{V}(q \wedge s) \vee (p \wedge s)\mathbf{V}(q \wedge s) \vee (q \wedge r)\mathbf{V}(q \wedge s)$;

**(A10-G)** Assume that $(\theta, N, L, w_i) \models \mathbf{G}(p \rightarrow \mathbf{X}p)$. The definitions of satisfaction of **G** and **X** imply (i) for any $w_j \in \mathsf{dom}\ L$ such that $L(w_i) \leq L(w_j)$, $(\theta, N, L, w_j) \models p$ implies $(\theta, N, L, w_k) \models p$ where $L(w_k) = L(w_j) + 1$. Also assume (ii) $(\theta, N, L, w_i) \models p$. By mathematical induction on $i$ using (ii) and (i), we infer that $(\theta, N, L, w_j) \models p$ for any $w_j \in \mathsf{dom}\ L$ such that $L(w_j) \leq L(w_i)$. Therefore, using the definition of satisfaction of **G** and **S3**, we conclude that $(\theta, N, L, w_i) \models \mathbf{G}(p \rightarrow \mathbf{X}p) \rightarrow (p \rightarrow \mathbf{G}p)$;

**(A12-Xbeg)** From **S6**, it is clear that (i) if $(\theta, N, L, w_j) \models \mathbf{beg}$ then $L(w_j) = 0$. Moreover, the definition of satisfaction of **X** says that (ii) $(\theta, N, L, w_i) \models \mathbf{X}(\mathbf{beg})$ implies $(\theta, N, L, w_j) \models \mathbf{beg}$ such that $L(w_j) = L(w_i) + 1$. Considering that (ii) implies (i), we reach a contradiction and conclude in this way that $(\theta, N, L, w_i) \models \neg\mathbf{X}(\mathbf{beg})$ due to **S2**;

**(A13-A)** According to **S8**, $(\theta, N, L, w_i) \models \mathbf{A}(p \rightarrow q)$ implies $(\theta, N, L_j, (L_j^{-1} \circ L)(w_i)) \models p \rightarrow q$ for any $L_j$ which agrees with $L$ on the state formulas satisfied up to $j$. Using **S3**, we can infer that $(\theta, N, L_k, (L_k^{-1} \circ L)(w_i)) \models p$ implies $(\theta, N, L_l, (L_l^{-1} \circ L)(w_i)) \models q$, moving in this way the quantification over behaviours to each instance of the satisfaction relation. Therefore, based on **S8** and **S3**, we infer $(\theta, N, L, w_i) \models \mathbf{A}(p \rightarrow q) \rightarrow (\mathbf{A}p \rightarrow \mathbf{A}q)$;

**(A15-EA)** Suppose that (i) $(\theta, N, L, w_i) \models \mathbf{E}p$ and (ii) $(\theta, N, L, w_i) \models \mathbf{AE}p$ is not true. From (i) and **S8**, we know that (iii) there is $L_j$ which agrees with $L$ on the state formulas satisfied up to $w_i$ such that $(\theta, N, L_j, (L_j^{-1} \circ L)(w_i)) \models p$. Moreover, (ii) and **S8** allow us to say that it is not the case that (iv) there is $L_k$ which agrees with $L$ on the state formulas satisfied up to $w_i$ such that $(\theta, N, L_k, (L_j^{-1} \circ L)(w_i)) \models \mathbf{E}p$. But (iv) and **S8** show that there is not an $L_k$ such that for every $L_l$ which agrees with $L_k$ on the formulas satisfied up to $(L_k^{-1} \circ L)(w_i)$ and $(\theta, N, L_l, (L_l^{-1} \circ L_k)((L_k^{-1} \circ L)(w_i))) \models p$, which contradicts (iii) because $(L_l^{-1} \circ L_k) \circ (L_k^{-1} \circ L) = L_l^{-1} \circ (L_k \circ L_k^{-1}) \circ L = L_l^{-1} \circ I \circ L = L_l^{-1} \circ L$. Therefore, applying **S3** to the negation of our assumption,we conclude that $(\theta, N, L, w_i) \models \mathbf{E}p \rightarrow \mathbf{AE}p$;

**(A16-EV)** Suppose that (i) $(\theta, N, L, w_i) \models (\mathbf{E}p)\mathbf{V}q$ but (ii) $(\theta, N, L, w_i) \models \mathbf{E}(p\mathbf{V}q)$ is not the case. From (i), **S7** and **S8**, (iii) there is $w_j \in \mathsf{dom}\ L$

with $L(w_i) < L(w_j)$ and $L_k$ which agrees with $L$ on the state formulas satisfied up to $w_j$ such that $(\theta, N, L_k, (L_k^{-1} \circ L)(w_j)) \models p$ and for every $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(w_k) < L(w_j)$, $(\theta, N, L, w_k) \models q$. In addition, from (ii), **S8** and **S7**, we see that for every $L_l$ which agrees with $L$ on the formulas satisfied up to $w_i$ and every $w_m \in \mathsf{dom}\ L_l$ with $L_l(w_i) < L_l(w_m)$, $(\theta, N, L_l, (L_l^{-1} \circ L)(w_m)) \models p$ and $(\theta, N, L_l, (L_l^{-1} \circ L)(w_n)) \models q$ for any $w_n \in \mathsf{dom}\ L_i$ where $L_l(w_i) < L_l(w_n) < L_i(w_m)$ are not both true. These hold for $L_l = L_k$ and $w_m = w_j$, but in this case (iii) is contradicted. Therefore, applying **S3** to the negation of our assumption, we conclude that $(\theta, N, L, w_i) \models (\mathbf{E}p)\mathbf{V}q \to \mathbf{E}(p\mathbf{V}q)$;

**(A17-AU)** Assume that $(\theta, N, L, w_i) \models \mathbf{A}(p \to \mathbf{X}(q\mathbf{U}p))$. By **S8**, **S3** and the definition of satisfaction of **X** and **U**, our assumption is easily shown to be equivalent to (i) for every $L_i$ which agrees with $L$ on the state formulas satisfied up to $w_i$, $(\theta, N, L_j, w_j) \models p$, where $w_k = (L_j^{-1} \circ L)(w_i)$, implies in the existence of an $w_l \in \mathsf{dom}\ L_j$ with $L_j(w_k) + 1 \leq L_j(w_l)$ such that $(\theta, N, L_j, w_l) \models p$ and $(\theta, N, L_j, w_m) \models q$ for any $w_m \in \mathsf{dom}\ L_i$ where $L_j(w_k) \leq L_j(w_m) < L_j(w_l)$. Assume in addition that (ii) $(\theta, N, L, w_i) \models p$. Note that (i) particularly holds for each $L_n$ which agrees with $L$ on the satisfied formulas up to and including $w_i$. In these cases, we can apply (ii) in (i) and infer that there is an $w_l' \in \mathsf{dom}\ L_n$ with $L_n(w_i) + 1 \leq L_n(w_l')$ such that $(\theta, N, L_n, w_l') \models p$, $(\theta, N, L_n, w_m') \models q$ for any $w_m' \in \mathsf{dom}\ L_n$ where $L_n(w_i) + 1 \leq L_n(w_m') < L_n(w_k')$. The definition of satisfaction of **U**, **X** and **S3** show that $(\theta, N, L, w_i) \models \mathbf{A}(p \to \mathbf{X}(q\mathbf{U}p)) \to (p \to \mathbf{XA}(q\mathbf{U}p))$;

**(A20-$\forall$)** Assume for $x \notin \mathit{Free}(p)$ that (i) $(\theta, N, L, w_i) \models \forall x \cdot p \to q$, and (ii) it is not the case that $(\theta, N, L, w_i) \models p \to \forall x \cdot q$. So, from (i), **S4**, **S3**, for every $v \in \mathsf{cod}\ N$ and every assignment $N_v$ for $\theta$ such that $N_v(y) = N(y)$ if $y \neq x$ or $N_v(y) = v$ otherwise, $(\theta, N_v, L, w_i) \models p$ implies $(\theta, N_v, L, w_i) \models q$. The consequent in this implication is also obtained from $(\theta, N, L, w_i) \models p$ in a structural induction argument, due to $x \notin \mathit{Free}(p)$. On the other hand, we have that $(\theta, N, L, w_i) \models p$ but $(\theta, N_v, L, w_i) \models q$ is not true for some $v$, $N_v$, due to (ii), **S4**, **S3**. Therefore, we reach a contradiction and conclude that $(\theta, N, L, w_i) \models \forall x \cdot (p \to q) \to (p \to \forall x \cdot q)$;

**(A22-EQ)** Assume that $(\theta, N, L, w_i) \models (t_1 = t_2)$. Consequently, for a given formula $p$ and any formula $q$, $(\theta, N, L, w_i) \models p\{q\backslash t_1\}$ implies $(\theta, N, L, w_i) \models p\{q\backslash t_2\}$. This is proved in detail by structural induction on the notions of interpretation and satisfaction based on the definition of substitution but is omitted here. Applying **S3** twice, we conclude that $(\theta, N, L, w_i) \models$

$(t_1 = t_2) \to (p\{q\backslash t_1\} \to p\{q\backslash t_2\})$;

**(A23-∃V)** Suppose that $x \notin Free(q)$ and (i) $(\theta, N, L, w_i) \models (\exists x \cdot p)\mathbf{V}q$ and (ii) $(\theta, N, L, w_i) \models \exists x \cdot (p)\mathbf{V}q$ is not the case. From (i), **S7** and **S4**, there is $w_j \in \mathsf{dom}\ L$ with $L(w_i) < L(w_j)$, $v \in \mathsf{cod}\ N$ and assignment $N_v$ with the usual definition such that $(\theta, N_v, L, w_j) \models p$ and for every $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(w_k) < L(w_j)$, $(\theta, N, L, w_k) \models q$. In addition, from (ii), **S4** and **S7**, for every $v' \in \mathsf{cod}\ N$, assignment $N_{v'}$ with the usual definition and every $w_l \in \mathsf{dom}\ L$ with $L(w_i) < L(w_l)$, $(\theta, N_{v'}, L, w_l) \models p$ and $(\theta, N_{v'}, L, w_m) \models q$ for any $w_m \in \mathsf{dom}\ L$ where $L(w_i) < L(w_m) < L(w_l)$ are not both true. Note that this is equivalent to universally quantifying $N_v$ only in the first half of the sentence because $x \notin Free(q)$. In particular (iii) holds for $N_{v'} = N_v$ and $w_l = w_j$ but in this case (ii) is contradicted. Therefore, applying **S3** to the negation of our assumption, we conclude that $(\theta, N, L, w_i) \models (\exists x \cdot p)\mathbf{V}q \to \exists x \cdot (p\mathbf{V}q)$;

**(A24-EQG)** Assume that $(\theta, N, L, w_j) \models (t_1 = t_2)$ for $t_1$, $t_2$ free from any attribute symbol. In particular, for any $w_i \in \mathsf{dom}\ L$ such that $L(w_j) \leq L(w_i)$, $(\theta, N, L, w_i) \models (t_1 = t_2)$, due to **S2**, $[\![t_1]\!]^{\theta,N}(w_i) = [\![t_1]\!]^{\theta,N}(w_j)$ and similarly for $t_2$. From the definition of satisfaction of **G** and **S3**, we conclude that $(\theta, N, L, w_i) \models (t_1 = t_2) \to \mathbf{G}(t_1 = t_2)$;

**(A26-∀A)** Assume that $(\theta, N, L, w_i) \models \forall x \cdot \mathbf{A}(p)$. According to **S4**, this means that for every $v \in \mathsf{cod}\ N$ and every assignment $N_v$ for $\theta$ such that $N_v(y) = N(y)$ if $y \neq x$ or $N_v(y) = v$ otherwise, $(\theta, N_v, L, w_i) \models \mathbf{A}p$. Now, from **S8**, we infer that for any $L_j$ which agrees with $L$ on the state formulas satisfied up to $w_i$, $(\theta, N_v, L_j, (L_j^{-1} \circ L)(w_i)) \models p$. Reversing the order of these universal quantifications over $N_v$ and $L_j$ and applying **S8**, **S4** and **S3** in this order, we conclude that $(\theta, N, L, w_i) \models \forall x \cdot \mathbf{A}p \to \mathbf{A}(\forall x \cdot p)$.

It remains to be shown that, if a set of sentences is related to a sentence by an entailment of $MSBTL$, then they are also related by the corresponding semantic consequence relation. That is, $\Psi \vdash_{\Delta} p$ implies $\Psi \models_{\Delta} p$. Not surprisingly, we use our proof calculus to decompose this problem. Assume that $\Psi \vdash_{\Delta} p$. The faithfulness condition in Definition 2.2.6 says that there is a derivation $(D, p)$ such that $(D, p) \in Pr_{\Delta}(\Psi, p)$. We proceed by structural induction:

BASE CASE: Characterised by derivations consisting of a single step, where $D = \{\,\}$. According to 2.2.6, we only need to examine the following case:

**$p \in Ax(\Delta)$:** Here, $\Psi = \{\,\}$. It is shown above that $(\theta, N, L, w_i) \models_{\Delta} p$ for any $\theta$, $N$, $L$ and $w_i \in \mathsf{dom}\ L$ whenever $p \in Ax(\Delta)$. Hence, $(\theta,N,L,$ $w_i) \models_\Delta q$ for any $q \in \Psi$ implies $(\theta, N, L, w_i) \models_\Delta p$, i.e., $\Psi \models_\Delta p$;

Note that there is no need to examine other base cases. For, if $p \in \Psi$, $\Psi \neq \{\,\}$ and then $D \neq \{\,\}$ due to the constraint in the faithfulness condition. If $p \notin Ax(\Delta) \cup \Psi$ and $D = \{\,\}$, $(D, p) \notin Pr_\Delta(\Psi, p)$ due to the minimality of $Pr_\Delta(\Psi, p)$.

INDUCTIVE STEP: Characterised by derivations constructed out of many steps. Since $(D, p) \in Pr_\Delta(\Psi, p)$ for some $D \neq \{\,\}$, this must be true because of the third case in the definition, due to the minimality of $Pr_\Delta(\Psi, p)$. This means that the following must be the case for some $\chi = \{(\delta_i, p_i) | \delta_i \cup \{p_i\} \subseteq \mathcal{G}(\Delta)\}$: (i) $D = \{(d_i, p_i) \in Pr_\Delta(\psi_i \cup \delta_i, p_i) | \psi_i \subseteq \Psi, \exists c \in \chi \cdot c = (\delta_i, p_i)\}$ and (ii) $\chi \Vdash_\Delta p$. Since our proof calculus is given in Hilbert-style, all the $\delta_i$s above have to be empty and can be ignored. By the inductive hypothesis, from (i) we obtain $\psi_i \models_\Delta p_i$ and because of $\cup\psi_i \subseteq \Psi$ and the monotonicity of $\models_\Delta$, we conclude (iii) $\Psi \models_\Delta p_i$. We showed above that each inference rule preserves validity. So, from (ii) we obtain (iv) $\{p_i | \exists\delta_i \cdot (\delta_i, p_i) \in \chi\} \models_\Delta p$. The transitivity of $\models_\Delta$ allows us to link (iii), (iv) and conclude that $\Psi \models_\Delta p$.

The above holds for any derivation $D$ such that $(D, p) \in Pr_\Delta(\Psi, p)$. Therefore, because of the faithfulness condition, we can conclude that $\Psi \vdash_\Delta^{MSBTL} p$ implies $\Psi \models_\Delta^{MSBTL} p$. ■ **(*MSBTL* Soundness)**

**Theorem 2.9.4** *MSBTL* is a logical system.

Proof: The signatures of *MSBTL* determine a category. The morphisms in this category are structure preserving in the sense that the components of each signature are mapped accordingly. The proof that $\mathbf{Sig}^{MSBTL}$ is a category is then developed in a way analogous to Theorem 2.3.2.

The entailment system of *MSBTL* is defined by a proof calculus where weakening and distributivity are logical. So, as shown in Section 2.3, the properties of reflexivity, monotonicity and transitivity of full entailment relations are automatically obtained as well as strong structurality due to the format of our axiom schemas. These ensure that *MSBTL* is a full entailment system.

We defined a model theory for *MSBTL* in Section 2.8. The collections of such models define categories as claimed in Proposition 2.9.1. In turn, they support an institution according to Lemma 2.9.2.

Entailment and institution obey a soundness condition according to Lemma 2.9.3, showing that *MSBTL* is a logic. Together with the proposed proof calculus, *MSBTL* constitutes a logical system. ■ (*MSBTL* **Logical System**)

The following result is of a negative nature:

**Theorem 2.9.5** *MSBTL* is not complete.

Proof: We prove this theorem in a way studied in detail by Abadi (1989), showing that the sentences true in the standard model of (Peano) arithmetic can be mapped into valid sentences of our temporal logic. Because arithmetical truth is undecidable, there is a true sentence such that neither itself nor its negation can be proved (Gödel 1931), this result is transfered to the temporal logic.

Consider a sentence $p$ written in the language of arithmetic presented in Figure 2.2. A (recursive) translation of $p$ into our temporal logic can be defined as $\iota(p) \overset{\text{def}}{=} p \wedge \bigwedge \{q | q \in Ax(\text{TA})\}$ where TA is the specification in Figure 2.16[13]. Note that $\iota(p)$ is well-defined because the language of PA is included in that of TA. Alternatively, we could have also chosen distinct symbol names and connected these theories from distinct logics by a functor, in which case our argument would be similar to the above.

We know that $\mathcal{A}_c = (\mathbb{N},\ 0_U,\ s_U,\ +_U,\ \times_U)$ is the standard model of theory PA. On the other hand, according to Definition 2.8.2, any model of TA must have the following structure: $\theta = (T,\ \mathcal{A}_t,\ \{N : \mathcal{N} \to \alpha\},\{\,\})$ for $\mathcal{A}_t = \{\mathcal{N}, (0_U, s_U, +_U, \times_U)\}$ and $T = (\alpha_0,\ \alpha,\ \rho,\ \Lambda)$. The rigid constant and function symbols specified through axioms (10.1) to (10.6) are interpreted as in the classical case because this set of axioms is the same of Figure 2.2. In addition, axiom (10.7) guarantees that, in any behaviour $L \in \Lambda$, each element of $\mathcal{N}$ denoted by the attribute symbol $n$ is the $N$-image of some $w \in \mathsf{dom}\ L$. Moreover, axiom (10.8) guarantees that each $w \in \mathsf{dom}\ L$ will be mapped to a unique element of $\mathcal{N}$ denoted by $n$. Therefore, $N$ is a bijection between $\mathsf{dom}\ L$ and $\mathcal{N}$. Because $L$ itself is a bijection between its domain and $\mathbb{N}$, $\mathcal{N}$ is isomorphic to $\mathbb{N}$. Picking $\alpha$ as any infinite set produces a model for TA. We infer in this way that $\mathcal{A}_c \models_{\text{PA}} p$ iff $\mathcal{A}_t \models_{\text{PA}} p$ iff $\theta \models_{\text{TA}} \iota(p)$, this last biconditional being justified by the definition of $\iota$ and the fact that sort symbols are rigid.

Suppose that TA $\models\ \iota(p)$ implies TA $\vdash\ \iota(p)$ for every sentence $p$. Applying the definition of our interpretation and the previous assumption to some $p$ such that $\mathcal{A}_c \models\ p$, this would mean that we have a method to determine whether or not a sentence is true in the standard model of arithmetic, but this contradicts the incompleteness result developed by Gödel. We have shown that *MSBTL* is incomplete. ■ (*MSBTL* **Incompleteness**)

[13]The idea behind this specification is to assign at each time instant the flexible symbol $n$ to a unique natural number and to establish in this way a bijection between the denotation of nat and $\mathbb{N}$. Axiom 10.7 says that in the beginning of time $n = 0$ and, for each element $x$ of sort nat, it will be assigned to $n$ eventually. Axiom 10.8 guarantees that the next value of $n$ is always the successor of its current value.

**Theory** TA
**sorts** $\mathsf{nat}$
**constants** $0 : \mathsf{nat}$
**operations** $s : \mathsf{nat} \to \mathsf{nat}; + : \mathsf{nat} \times \mathsf{nat} \to \mathsf{nat}; * : \mathsf{nat} \times \mathsf{nat} \to \mathsf{nat}$
**attributes** $n : \mathsf{nat}$
**axioms** $x, y : \mathsf{nat}$[14]

$$\neg(0 = s(x)) \tag{10.1}$$
$$s(x) = s(y) \to x = y \tag{10.2}$$
$$x + 0 = x \tag{10.3}$$
$$x + s(y) = s(x + y) \tag{10.4}$$
$$x * 0 = 0 \tag{10.5}$$
$$x * s(y) = x * y + x \tag{10.6}$$
$$\mathbf{beg} \to n = 0 \land \mathbf{F}(n = x) \tag{10.7}$$
$$n = x \to \mathbf{X}(n = s(x)) \tag{10.8}$$

**End**

Figure 2.16: Temporal first-order theory of Peano arithmetic.

It is important to stress that the negative result above only holds for the interpretation structures we have chosen here. It may be possible to find a slightly different semantics for our logic so that it becomes complete. Andréka *et al.* (1995) have applied Correspondence Theory as proposed by van Benthem (1984) to map first-order temporal logic into classical logic, which is complete. Thus this result may be transfered to the temporal framework. In our case, it appears to be necessary to study in detail first if the propositional fragment of the logic above is medium complete before proceeding with the study of the first-order framework. Research in this direction is under way.

Concluding this section, we return to Proposition 2.7.4. We show that the inference rule proposed therein is admissible in $MSBTL$. As a corollary, we deduce that a logical system is obtained as a result of adding such rule to the proof calculus of $MSBTL$.

**Theorem 2.9.6 (Admissibility of INTRO)** Assume that $s \in Sort(\Delta)$, $t \in Attr(\Delta)$ with $type(t) = \epsilon \to s$ and $\prec \in Act(\Delta)$ with $type(\prec) = s \times s$ for a given $\Delta$ in $\mathsf{obj}\ \mathbf{Sig}^{MSTBL}$. The following inference rule is admissible in $MSBTL$:

$$\boxed{(\mathbf{INTRO})\quad \dfrac{\begin{array}{llll} 1. & \mathbf{IRR} & 4. & \mathbf{APROG} \\ 2. & \mathbf{STAB} & 5. & \mathbf{FREE} \\ 3. & \mathbf{TERM} & 6. & \mathbf{LIM} \end{array}}{\forall x \cdot (\mathbf{F}(p[x]) \to \exists y \cdot \mathbf{F}(y \prec x \land p[y])) \to \forall x \cdot \neg \mathbf{F}(p[x])}}$$

[14]The variables $x$ and $y$ of sort $\mathsf{nat}$ appear implicitly quantified in the subsequent axioms.

Proof: We assume given a signature $\Delta$ such that $s \in Sort(\Delta)$, $t \in Attr(\Delta)$ with $type(t) = \epsilon \to s$ and $\prec \in Act(\Delta)$ with $type(\prec) = s \times s$. We write $x \prec y$ for $\prec (x, y)$. A classification $Class(\Delta)$ for $\Delta$ is also assumed to exist. We have to show that, for any frame $T = (\alpha,\ \alpha_0,\ \rho,\ \Lambda)$, assuming that the premises of **INTRO** are valid in $T$, the conclusion is also valid in $T$.

From (2), it is easy to see by applying the rule of temporal generalisation **R2-G** that $\mathbf{FG}(\forall x, y \cdot x \prec y \to \mathbf{X}(x \prec y))$ is the case. Conjoining this sentence to (3) and relying on the distributivity of both **FG** and $\forall$ over conjunction, we conclude that the symbol $\prec$ will eventually have a rigid interpretation in any model $\theta$ based on $T$. We call the world from which this becomes true as $w_0 \in \mathsf{dom}\ L$, for each $L \in \Lambda$.

From (4), it is not difficult to derive the following sentence:

$$\mathbf{G}(\forall x \cdot t = x \to \mathbf{X}(t = x \vee t \prec x)) \to \exists x \cdot \mathbf{FG}(t = x) \tag{2.9.4}$$

Suppose that for some model $\theta = (T,\ U,\ G,\ A)$, there is an infinitely decreasing sequence of values from $s_U$, $S = \langle t_0, t_1, \ldots \rangle$, which are related by $\prec_U$. We are going to show based on $S$ that $\theta$ cannot exist while satisfying the premises of **INTRO** and (2.9.4) in particular.

Since our assumption guarantees that (1) is true in $\theta$, we can infer that (a) $t_i \neq t_{i+1}$, for every $i \geq 0$. Moreover, from (5), for every world $w_i$ of a fixed behaviour $L_i$ such that $L_i(w_0) < L_i(w_i)$ and $[\![t]\!]^{\theta,N}(w_i) = t_{L_i(w_i)-L_i(w_0)-1}$, we know that there is another behaviour $L_{i+1}$ with a past history up to $s_i$ equivalent to that of $L_i$ such that for $w_{i+1} \in \mathsf{dom}\ L_{i+1}$ with $L_{i+1}(s_{i+1}) = L_i(w_i) + 1$, $[\![t]\!]^{\theta,N}(w_{i+1}) = t_{L_{i+1}(w_{i+1})-L_i(w_0)-1}$. This shows that the antecedent of (6) is satisfied, so we also obtain based on the discreteness of $S$ and $L$ that (b) there is an $L \in \Lambda$ such that $\forall s_i \in \mathsf{dom}\ L \cdot L(w_0) < L(s_i) \to [\![t]\!]^{\theta,N}(w_i) = t_{L(w_i)-L(w_0)-1}$ for any assignment $N$ for $Class(\Delta)$. The antecedent of (2.9.4) is satisfied from $s_1$ onwards due to (b) and the rigid character of $\prec_U$, but the consequent of this implication is never obtained in $L$ since the value of $t$ keeps changing forever, according to (a) and (b). In this way, our assumption of an infinitely decreasing chain of values from $s_U$ related by $\prec_U$ generates a contradiction. Therefore, there is no such an infinite sequence in any $\theta$ based on $T$.

Suppose that the antecedent of the conclusion is the case but the consequent is not. From the latter, we know that (i) there is at least one element in $s_U$ for any $\theta$ based on $T$ containing $U$. From the former, we know that (ii) for every $t_i \in s_U$, there is a $t_{i+1} \in s_U$ such that $t_{i+1} \prec_U t_i$. From (i) and (ii), we can infer that there is an infinitely decreasing sequence of values from $s_U$ related by $\prec_U$, but this is a contradiction. Hence, the conclusion of **INTRO** is valid in $T$. We conclude that **INTRO** is admissible. ■ **(Admissibility of INTRO)**

**Corolary 2.9.7 (Soundness of $MSBTL^+$)** $MSBTL^+$ is sound.

Proof: Based on Lemma 2.9.3, we only need to show that **INTRO** preserves validity. But this is precisely what Theorem 2.9.6 states. Therefore, $MSBTL^+$ is sound. ■ ($MSBTL^+$ **Soundness)**

## 2.10 Summary and Related Work

We began this chapter arguing in favour of a proof-theoretic approach to rigorous software development. The many steps of the development process were examined and the benefits of adopting this viewpoint were outlined. A number of authors, such as Lehman *et al.* (1984), Turski and Maibaum (1987), Fiadeiro *et al.* (1991) and de Queiroz (1990) appear to share a similar view, which is not original to our work.

Afterwards, we presented definitions of many general logical structures which appear to provide an adequate foundation for our proof-theoretic studies. In particular, we stressed the fundamental role of category theory as a means of developing a software development theory relatively independent from the adopted logical system as well as of facilitating the transposition of results between related systems. General logics have been extensively studied in the literature by Fiadeiro and Sernadas (1988), Goguen and Burstall (1992), Meseguer (1989), Fiadeiro and Maibaum (1993) and Cerioli and Meseguer (1997) among others. On top of these studies we have introduced a minor but nevertheless necessary assumption of syntactic vocabulary closure and a practical definition of proof-calculus which seems to be a good alternative if compared to the complicated categorical definition adopted by Meseguer (1989).

A series of entailment systems was subsequently defined culminating in the introduction of a new many-sorted, first-order, branching time logical system with equality, which appears to be adequate for designing extensible software systems. We examined in detail the proof-theory of fragments of this system based on their Hilbert-style definitions, providing realistic application examples, and finally assessed other characteristics also related to their model-theory, namely soundness and completeness. From the proof-theoretic side, it is unfortunate to have only a Hilbert-style proof-calculus for our connectives because more elegant proof-theoretic techniques such as cut elimination cannot be effectively applied in this way. From the model-theoretic side, we discovered that the adopted semantics does not yield a completeness result even though this may be possible in a slightly changed framework. These limitations do not, however, preclude the practical application of our logical system and in fact it is not yet clear if the solution of these problems may lead to the development of a useful framework.

Perhaps the major contribution of this chapter is the proposed logical system and the corresponding design principles developed to support the specification and verification of software systems. A substantial number of related formalisms with their own principles has already appeared in the literature. Chandy and Misra (1988) have developed UNITY, which is not strictly speaking a temporal logical system but supports the design of concurrent systems. UNITY lacks an elegant treatment of naming, which is resolved in terms of set theoretic operations on the symbols of each presentation and does not support in this way modularised design, since name clashes may occur in combining specifications which were developed in isolation. This is treated here by the categorical constructions adopted following Fiadeiro and Maibaum (1992). Among temporal logical formalisms, $TLA$ (Lamport 1994) and the linear time logic proposed by Manna and Pnueli (1989) are close to ours, although they were not developed with the same assumptions in mind and thus do not provide a proof-theoretic account to each logical symbol, as the enabledness connective demonstrates. We believe that such a kind of definition is fundamental in rigorous software development and, in particular, when automating the process. Another related logic is $CTL^*$ (Emerson 1990), which is a propositional branching time logic where the branching connective has a slightly distinct meaning. We have provided both a functor showing how to interpret $CTL^*$ theories into our formalism and the rationale justifying the choice of a distinct modality meaning. Concerning the design principles proposed here in the form of derived inference rules (apart from the adopted categorical constructions), the anchored induction rule appears to be quite a standard way of dealing with the verification of safety properties. The lattice rule, on the other hand, normally lacks either methodological guidance or an axiomatic basis upon which it can be applied. We are only aware of other works herein both problems are treated just for the case of rigid relation symbols in terms of non standard methods. We shall have the opportunity to exemplify the application of our principles throughout the following chapters.

# Chapter 3

# Designing Open Reconfigurable Systems

Distributed systems have provided one of the most pertinent frameworks for organising separate independently produced software artifacts. Essentially, a *distributed system* consists in a set of loosely interconnected software *components.* For instance, a set of procedures put together to run as a sequential program cannot be regarded as a distributed system unless there are methods supporting the replacement of some of these components at run time and also their execution in separate address spaces. Clearly then, the definition above is not very informative and has to be complemented by a model which specifies how components are connected to each other and interact among themselves.

Distributed system models come in different flavours. First of all, one needs to consider whether *interaction* is to be supported by shared or isolated entities. *Shared memory* allows distinct components to have read and perhaps write access to a common storage. This is peculiar to the development of protocols for ensuring distributed memory consistency (Raynal and Mizzymo 1993). *Shared control* allows distinct components to observe the same event simultaneously. A family of so-called *coordination languages* is based on this notion (Ciancarini and Hankin 1996). Interaction based on sharing is necessarily synchronous. On the other hand, *message passing* is not necessarily so. Messages are transmitted in *asynchronous* mode if and only if it is not possible to place internal bounds on communication delays nor on the relative speed of each component. Otherwise, the mode of interaction is considered to be *synchronous.* In models based on message passing, wherein interaction is directed, there is also the issue of deciding the number of *participants* allowed in each interaction. If there must be only one *recipient* for each message, we say *point-to-point* communication is supported. At the opposite extreme, *broadcasting* is characterised by the fact that each message is always distributed to all the components of the system.

Distributed systems based on these models may support extensibility in an effective manner if they are also open and reconfigurable. A distributed system is said to be *reconfigurable* if and only if the *interconnection topology* of its components, or more simply its *configuration*, may vary with time. Moreover, the system is said to be *open* whenever it may eventually interact with an environment over which little if any control is retained. Because few assumptions can be made about the environment and the dynamic configuration of the system, it becomes easier to support changes which lead to extended functionality or structure. As we argued in the introduction, since openness and reconfigurability seem to enforce extensibility, it appears to be reasonable to anticipate their use and introduce explicit support to these notions at more abstract levels of the development process such as when performing software design.

Designing open reconfigurable distributed systems in a rigorous way does not appear to be an easy task. For example, Abadi and Lamport (1994) adopted the temporal logic $TLA$ in a rely-guarantee style to deal with openness, but left reconfigurability completely untreated. On the other hand, a whole field of study was uncovered when Milner *et al.* (1992) proposed a synchronous value passing process calculus in which names are primitive and can be passed around to allow the respective processes to reconfigure. However, they have preferred to leave the notion of openness untouched. Both notions were addressed by Agha *et al.* (1994) in terms of the so-called *actor model*, which is based on asynchronous message passing, but at a level of abstraction very close to implementation and without concern for rigorous verification of properties.

A model of distributed systems may be expressively rich enough to capture openness, reconfigurability and other notions that support extensibility. We believe this to be the case of the actor model. This is why we study in this chapter how to provide explicit support for this model using a proof calculus that extends our work of Chapter 2. Manna and Pnueli (1983) have also applied, at lower levels of abstraction, this idea of particularising a temporal logical system. In particular, we follow the terminology proposed by Fiadeiro *et al.* (1991) and claim to give a temporal proof-theoretic semantics for the interaction primitives of the actor model. We provide methods and principles to support specifying, composing and reasoning about actor communities. We also show that other message passing modes of interaction and other notions supporting extensibility may be treated in terms of actors. This open reconfigurable systems design initiative based on actors initiated by Duarte (1997b) is indeed possible due to a result of Koymans (1987), who first showed that by adopting purpose built temporal logics one can treat a variety of message passing modes of interaction.

We proceed by introducing the actor model and discussing some relevant issues in the design of the respective temporal proof-calculus. Subsequently, we describe our approach to the specification and verification of actor systems, illustrating the technicalities involved by means of a simple example. A summary and a comparison with related work appear in the last section.

## 3.1 Issues in the Design of a Proof Theory for the Actor Model

Since the pioneering work of Hewitt and Baker (1977) on the foundations of concurrency, a promising model of open distributed systems has been developed, initially by Clinger (1981) and lately by Agha (1986), Talcott (1996b) and others. The so-called actor model regards distributed systems as communities of objects with encapsulated state which may only be changed by performing local computations. Message passing between actors is buffered, point-to-point and asynchronous, based on a localised naming scheme. As a result of processing messages, new concurrent actors can be created, local computations can occur and actor names can be communicated.

Considering the characteristics above, it seems to be a natural research direction to abstract from previous work in which the model was realised in diverse programming languages and semantic domains in order to examine the step-by-step development, and here in particular the design, of open reconfigurable systems in terms of actor communities. Agha (1986) identified the basic primitives required to support the model and outlined a generic operational semantics for actor languages. In (Agha *et al.* 1997), the operational semantics of a complete language was developed along with criteria for dynamically composing interacting actor components. Alternative semantic domains defined in terms of the inference rules of rewriting and linear logic were studied by Talcott (1996a), Darlington and Guo (1995), respectively. All these works have focused on describing in an operational manner the behaviour of actor systems.

In Chapter 2, we defined a logical system which appears to be expressive enough to support the design of actor systems. State and change, for instance, can be represented by sets of attribute and action symbols. Moreover, creation and naming may be dealt with in the usual way studied by Ehrich *et al.* (1988), in terms of the first-order features of the logic. This is to say, a distinguished sort symbol denoting object names is considered to be part of every signature and all the attribute and action symbols are regarded to be parameterised by the respective sort, extending the originally provided specifications. To avoid conflicts between the creation of new actors and the satisfiability of Barcan formulas, every actor specification may carry an auxiliary existential boolean attribute symbol. According to this approach, objects that have not been created, i.e., their respective attribute is equal to false, do not play any role, paraphrasing America and de Boer (1996).

Concerning the buffered, point-to-point, asynchronous mode of interaction between actors, a faithful approximation can be defined by introducing another set of logical symbols in each specification and providing an extended axiomatisation which depends on these new symbols. In particular, because the actor model requires the delivery and consumption of a message to be guaranteed whenever it becomes possible often enough for the target actor to deliver such a functionality, *fairness* requirements which demand specifying when these events may occur as it is impossible to determine *a priori* how the environment will evolve, the full expressiveness of our branching time logic has to be used.

Considering this rationale, actor specifications should look like Figure 3.1. Therein, buffer cells are specified which dynamically allocate a new cell for each stored integer. Attribute symbols represent the actor state whereas messages and local computations are represented by action symbols. The connectives **E**, **X**, **F** are as previously defined. In Axiom (11.9), for instance, **X** is used to state that, if a message $put(v)$ is consumed by the last cell of the buffer ($lst = \mathrm{T}$), in the next instant another cell containing the value $v$ will be created and linked to the current one ($\mathbf{new}(\ item, n, v) \wedge link(n)$). Subsequently, the buffer will have reconfigured accordingly. On the other hand, $\leftarrow$ is a new definable temporal connective which is required in stating that a property holds only if preceded by the occurrence of another property. Axiom (11.13) determines that neither of the two events above happen unless the appropriate cell consumes a *put* message first. We shall continue to explain this example in the following sections.

## 3.2 An Axiomatisation of the Actor Model

### 3.2.1 Representing Actors

We use theory signatures to define the symbols that can be used in writing each specification. Specifications, in turn, consist of finite sets of axioms defining theory presentations. Both notions are the same as explained in Chapter 2, but here we particularise even further the structure of $MSBTL$ signatures to cater for the peculiarities of the actor model. We also use a shorthand notation which will facilitate the exposition of the formalism. Theory signatures for actor specification are defined as follows:

**Actor** BUFFERCELL
**data types** $\mathsf{addr}, \mathsf{bool}, \mathsf{int}$ $(\mathrm{T}, \mathrm{F} : \mathsf{bool})$
**attributes** $val : \mathsf{int}; nxt : \mathsf{addr}; void, lst, up : \mathsf{bool}$
**actions** $nil, item(\mathsf{int})$ : **local + extrn birth**;
$go, cons, link(\mathsf{addr})$ : **local computation**;
$put(\mathsf{int}), get(\mathsf{addr})$ : **local + extrn message**;
$reply(\mathsf{int})$ : **extrn message**
**axioms** $k, n : \mathsf{addr}; v : \mathsf{int}; x, y : \mathsf{bool}$

$nil \rightarrow void = \mathrm{T} \wedge lst = \mathrm{T} \wedge up = \mathrm{F}$ (11.1)

$item(v) \rightarrow val = v \wedge void = \mathrm{F} \wedge lst = \mathrm{T} \wedge up = \mathrm{F}$ (11.2)

$nil \vee item(v) \rightarrow \mathbf{X}(go)$ (11.3)

$go \rightarrow \mathbf{X}(up = \mathrm{T})$ (11.4)

$go \wedge val = v \wedge void = x \wedge nxt = n \wedge lst = y \rightarrow \mathbf{X}(val = v \wedge void = x \wedge nxt = n \wedge lst = y)$ (11.5)

$cons \wedge nxt = n \wedge lst = x \wedge up = y \rightarrow \mathbf{X}(void = \mathrm{T} \wedge nxt = n \wedge lst = x \wedge up = y)$ (11.6)

$link(n) \rightarrow \mathbf{X}(nxt = n \wedge lst = \mathrm{F})$ (11.7)

$link(n) \wedge val = v \wedge void = x \wedge up = y \rightarrow \mathbf{X}(val = v \wedge void = x \wedge up = y)$ (11.8)

$put(v) \wedge lst = \mathrm{T} \rightarrow \mathbf{X}(\exists n \cdot \mathbf{new}(item, n, v) \wedge link(n))$ (11.9)

$put(v) \wedge lst = \mathrm{F} \wedge nxt = n \rightarrow \mathbf{X}(\mathbf{send}\ put, n, v\ ())$ (11.10)

$get(n) \wedge void = \mathrm{F} \wedge val = v \rightarrow \mathbf{X}(\mathbf{send}\ reply, n, v\ (\wedge) cons)$ (11.11)

$get(n) \wedge void = \mathrm{T} \wedge lst = \mathrm{F} \wedge nxt = k \rightarrow \mathbf{X}(\mathbf{send}\ get, k, n\ ())$ (11.12)

$\exists n \cdot \mathbf{new}(item, n, v) \vee link(n) \leftarrow put(v) \wedge lst = \mathrm{T}$ (11.13)

$\mathbf{send}\ reply, n, v\ (\vee) cons \leftarrow get(n) \wedge val = v \wedge void = \mathrm{F}$ (11.14)

$\mathbf{send}\ put, k, v\ (\leftarrow) put(v) \wedge nxt = k \wedge lst = \mathrm{F}$ (11.15)

$\mathbf{send}\ get, k, n\ (\leftarrow) get(n) \wedge nxt = k \wedge void = \mathrm{T} \wedge lst = \mathrm{F}$ (11.16)

$up = \mathrm{T} \rightarrow \mathbf{FE}(\mathbf{deliv}\ (put, v)) \wedge \mathbf{FE}(put(v)) \wedge \mathbf{FE}(\mathbf{deliv}\ (get, n)) \wedge \mathbf{FE}(get(n))$ (11.17)

**End**

Figure 3.1: Specification of integer buffer cells.

**Definition 3.2.1 (Actor Signature)** An *actor signature* $\Delta = (\Sigma, \mathcal{A}, \Gamma)$ is a triple of disjoint and finite families of symbols such that:

- $\Sigma = (S, \Omega)$ is a universe signature, i.e., $S$ is a set of rigid sort symbols and $\Omega$ is an $S^*_{fin} \times S$-indexed family of rigid function symbols[1]. We also require that $\mathsf{addr} \in S$, representing the sort of mail addresses (or actor names);
- $\mathcal{A}$ (or $\mathcal{A}_l$) is an $S^*_{fin} \times S$-indexed family of flexible attribute symbols;
- $\Gamma = (\Gamma_e, \Gamma_l, \Gamma_c)$ is a triple of $S^*_{fin}$-indexed families of action symbols such that $(\Gamma_e \cup \Gamma_l) \cap \Gamma_c = \{\,\}$. $\Gamma_c$ is a set of local computation symbols. The elements of $\Gamma_e$ and $\Gamma_l$ represent, respectively, events to be requested from the environment and provided locally[2]. Each of these two sets contains distinguished sub-sets of message and birth symbols, e.g. $\Gamma_l - \Gamma_{l_b}$ and $\Gamma_{l_b}$.

We write $\epsilon \rightarrow s$-indexed families of signature symbols as if $s$ were their single index. Given a set or sequence of such symbols $X$, we write as $X_{\langle s_1, \ldots, s_n \rangle, s}$ the sub-

[1]We usually consider that the enumerated constants are all different from each other.
[2]Because actors may self-address requests, $\Gamma_e$ and $\Gamma_l$ should not be disjoint in general.

set or sub-sequence of $X$ containing symbols of type $\langle s_1, \ldots, s_n \rangle \rightarrow s$ only. To make reference to specific sets of signature symbols, we operate with subscripts to denote operations on sub-sets. For instance, $\Gamma_{e_b} \cap \Gamma_{l_b}$ is written as $\Gamma_{e_b \cap l_b}$. $\square$

In the example specification of Figure 3.1, addr, bool and int are the sort symbols that constitute, together with their implicitly specified constants and operations, the universe signature $\Sigma$. Clearly, the sort of mail addresses addr has to be part of every signature. Otherwise, some specified actors would be useless without the ability of exchanging messages or creating new actors. Still in the example, $val$ (current value), $nxt$ (next cell address), $void$ (consumed content), $lst$ (last cell) and $up$ (live cell) are the attribute symbols in $\mathcal{A}$. In the particular terminology of the actor model, they are called acquaintances, which may be determined at creation time or in performing local computations.

The structure of the set of action symbols differs from those of Sernadas *et al.* (1995), Fiadeiro and Maibaum (1992), who advocate similar logics, and also from our definitions in the previous chapter. Each actor specification may guarantee the occurrence of externally required events and may determine that the occurrence of some events is required from the environment. Actor specifications may also define local computations. Because of these distinctions, the set of action symbols is divided into $\Gamma_l$, $\Gamma_e$ and $\Gamma_c$, respectively. The first two of these are partitioned into sub-sets of symbols to represent messages and births, $\Gamma_{e-e_b}$ and $\Gamma_{e_b}$ for instance. Actors interact via asynchronously transmitted messages, denoted by the symbols in $\Gamma_{(l-l_b)\cup(e-e_b)}$, which are used in many different ways. For instance, $put(v)$ represents the consumption of a message $put$ carrying $v$ as its contents and **send** $put, n, v$ () specifies that the same message and contents are transmitted to an object whose mail address is $n$. The distinguished uses of signature symbols also apply to the creation of actors, through the primitive **new** and the subsequent occurrence of birth actions in $\Gamma_{l_b \cup e_b}$. All these events can only occur carrying a finite number of acquaintances and are exemplified by the action symbols in Figure 3.1.

As is usual in a proof-theoretic approach, cf. Fiadeiro *et al.* (1991), Wieringa *et al.* (1995), we extend signatures with some new logical symbols. The situation here resembles the use of hidden symbols in algebraic specifications (Ehrig and Mahr 1985). Therein, the specifier may need to use an externally unavailable language to specify complex data types. Herein, we use a simpler language to specify complex patterns of behaviour presented by every actor, defined in terms of a more complex language. This extended language will be used to provide an implicit proof-theoretic semantics for the actor primitives and that is why it should not be required from the specifier of each signature.

**Definition 3.2.2 (Extended Actor Signature)** Given an actor signature $\Delta = (\Sigma, \mathcal{A}_l, \Gamma)$ such that $\Sigma = (S, \Omega)$ and $\Gamma = (\Gamma_e, \Gamma_l, \Gamma_c)$, the triple $\lambda\Delta = (\lambda\Sigma, \lambda\mathcal{A}, \lambda\Gamma)$ is said to be the *extended signature* of $\Delta$ if and only if:

1. $\lambda\Sigma = (S \cup \{\mathsf{bool}\}, \Omega \cup \{\mathrm{T}_{\mathsf{bool}}, \mathrm{F}_{\mathsf{bool}}, \mathrm{NOT}_{\mathsf{bool}\to\mathsf{bool}}\})$;

2. $\lambda\mathcal{A} = (\mathcal{A}_l, \mathcal{A}_i, \mathcal{A}_s, \mathcal{A}_d)$, such that (i) for each $c \in \Gamma_{l_b}$ of sort $\langle s_1, \ldots, s_n\rangle$ there is an $init_c \in \mathcal{A}_{i_{\langle s_1,\ldots,s_n\rangle,\mathsf{bool}}}$; (ii) for each $c \in \Gamma_{(e-e_b)\cup(l-l_b)}$ of sort $\langle s_1, \ldots, s_n\rangle$ there is a $sent_c \in \mathcal{A}_{s_{\langle s_1,\ldots,s_n\rangle,\mathsf{bool}}}$, and (iii) for each $c \in \Gamma_{l-l_b}$ of sort $\langle s_1, \ldots, s_n\rangle$ there is a $delivd_c \in \mathcal{A}_{d_{\langle s_1,\ldots,s_n\rangle,\mathsf{bool}}}$. All the symbols in the respective components of $\lambda\mathcal{A}$ are due to (i), (ii) and (iii);

3. $\lambda\Gamma = (\Gamma_e, \Gamma_{out}, \Gamma_l, \Gamma_{in}, \Gamma_c, \Gamma_{rcv})$, where (i) for each $c \in \Gamma_e$ of sort $\langle s_1, \ldots, s_n\rangle$ there is an $out_c \in \Gamma_{out_{\langle \mathsf{addr},\mathsf{addr},s_1,\ldots,s_n\rangle}}$; (ii) for each $c \in \Gamma_l$ of sort $\langle s_1, \ldots, s_n\rangle$ there is an $in_c \in \Gamma_{in_{\langle \mathsf{addr},\mathsf{addr},s_1,\ldots,s_n\rangle}}$, and (iii) for each $c \in \Gamma_{l-l_b}$ of sort $\langle s_1, \ldots, s_n\rangle$ there is a $rcv_c \in \Gamma_{rcv_{\langle s_1,\ldots,s_n\rangle}}$ such that $\Gamma_{(in\cup out)\cap rcv} = \{\,\}$ and that $in_c = out_c$ if and only if $c \in \Gamma_{e\cap l}$. All the symbols in the respective components of $\lambda\Gamma$ are due to (i), (ii) and (iii). □

That is to say, the original universe signature is extended with a boolean sort symbol, new attribute symbols are provided to deal with the existence of actors and buffering of messages, and new action symbols are introduced to handle creation and interaction. Hereafter, we will not make any distinction between extended signatures and actor signatures.

A central feature of actors is interaction. Here, it is simulated using the action symbols $out_c$ and $in_d$ which happen simultaneously for any $c \in \Gamma_e$ and $d \in \Gamma'_l$ belonging to the actor communities, populations of objects complying with the same specification, requesting and providing the event respectively. These symbols correspond either to the dispatch of a message or the request of an actor birth. The occurrence of these logical actions plays the role of the interaction steps of Talcott (1996b). For an interaction represented by $c$ between actors of the same community, hence required and provided locally and member of $\Gamma_{e\cap l}$, the occurrence of the new actions above is obliged to be synchronous by the second constraint in (3.iii) of Definition 3.2.2. Otherwise, this synchronisation must be supported by the existence of a morphism identifying these symbols as shared by the distinct signatures, as discussed in Section 3.4. Asynchrony in message transmission is guaranteed by forcing $out_c|in_d$ to happen strictly before $rcv_d$, which in turn has to occur strictly before $d$ itself. The two last symbols correspond to the occurrence of the delivery and consumption of the message, respectively. Finally, (double) buffering is captured by the attribute

$delivd_d$ ($sent_c$) becoming true for some values whenever these values are delivered (sent) in a message. Of course, these new symbols do not explicitly appear in specifications but their behavioural constraints will have to be captured by our axiomatisation. Also, according to the definition above, ill formed messages are not allowed — as action symbols, messages always have a locally correct representation at the sender — and dispatched messages which do not belong to the language available to the target actor are never delivered.

Following America and de Boer (1996), we consider that in a given point in time it is only possible to deal with the existing actors at that moment. Accordingly, an object will have some $init_c$ attribute equalised to T(RUE) for some sequence of terms $\vec{v_c}$ only if the occurrence of an action $in_c(\vec{v_c})$, $c \in \Gamma_{l_b}$, gives rise to its birth. The structure of communities of actors which comply with the same specification, each of which having a distinguished mail address, is defined below:

**Definition 3.2.3 (Actor Community Signature)** Given a signature $\Delta = (\Sigma, \mathcal{A}, \Gamma)$, a *community signature* $\Delta^{\mathsf{P}}$ is obtained by "parameterising" $\Delta$ with sort $\mathsf{P}$. That is, $\Sigma^{\mathsf{P}} \overset{\text{def}}{=} (S \cup \{\mathsf{P}\}, \Omega)$; $\mathcal{A}^{\mathsf{P}}$ is obtained from $\mathcal{A}$ by adding the parameter sort $\mathsf{P}$ to each of its attribute symbols; and $\Gamma^{\mathsf{P}}$ is obtained from $\Gamma$ by adding the parameter sort $\mathsf{P}$ to each action symbol in $\Gamma_e$, $\Gamma_l$, $\Gamma_c$ and $\Gamma_{rcv}$. The other symbols of $\Delta$ remain the same in $\Delta^{\mathsf{P}}$. □

Clearly, the parameter sort $\mathsf{P}$ of every community should be $\mathsf{addr}$. Indeed, as identified by Talcott (1996b), actor semantics should be parameterised by sets of actor addresses. Due to our definition, a new argument is added to the appropriate signature symbols and its instances will be actor names. In this way, the basic operations on object references identified by America and de Boer (1996), *equality test* and *dereferencing*, are supported. However, signatures alone do not support a modular design discipline, obliging the entire structure of complex systems to be represented as single entities. The required means of composition shall be studied in Section 3.4.

Due to the parameterisation of signatures by $\mathsf{addr}$, we are allowed to adopt the usual object-based notation of prefixing the name of an object to the logical expressions pertaining to it. In this way, we can move parameters outwards and write $p(n, \vec{v_p})$ as $n.p(\vec{v_p})$ for any attribute and action symbol $p$. This lifts in a compositional manner to all the expressions in each language. Adopting this convention, for each pair of formulas $p$ and $q$, say, we have $n.p \wedge n.q \equiv n.(p \wedge q)$. Sentences of this kind are called *global* as opposed to the *local* ones which have the focus actor striped out. Assuming that $n$, $n_i \in Term(\Delta)_{\mathsf{addr}}$, the usual actor primitives defined below are also admissible in specifications and proofs:

| For | in | formula | reads | represents |
|---|---|---|---|---|
| — | — | $n.\mathbf{init}$ | initialisation | $\bigvee\{\exists\vec{v_c}\cdot c(n,\vec{v_c})\vert c\in\Gamma_{l_b}\}$ |
| $\vec{v_c}$ | $Term(\Delta)$ | $n_1.\mathbf{new}(c,n_2,\vec{v_c})$ | actor creation | $out_c(n_1,n_2,\vec{v_c}),$ if $c\in\Gamma_{e_b}$ <br> $in_c(n_1,n_2,\vec{v_c}),$ if $c\in\Gamma_{l_b}$ |
| $\vec{v_c}$ | $Term(\Delta)$ | $n_1.\mathbf{send}\ c,n_2,\vec{v_c}\ ()$ | message dispatch | $out_c(n_1,n_2,\vec{v_c}),$ if $c\in\Gamma_{e-e_b}$ <br> $in_c(n_1,n_2,\vec{v_c}),$ if $c\in\Gamma_{l-l_b}$ |
| $\vec{v_c}$ | $Term(\Delta)$ | $n.\mathbf{deliv}\ (c,\vec{v_c})$ | message delivery | $rcv_c(n,\vec{v_c}),$ if $c\in\Gamma_{l-l_b}$ |

To deal with our examples in a more effective way, we also adopt the following definitions of not so standard temporal connectives of strict precedence:

**(D14-IP)** $q_1 \overset{i}{\leftarrow}_p q_2 \overset{\text{def}}{=} p \to (\neg q_1)\mathbf{W}(q_2 \wedge \neg q_1)$;

**(D15-P)** $q_1 \leftarrow_p q_2 \overset{\text{def}}{=} q_1 \overset{i}{\leftarrow}_p q_2 \wedge (q_1 \to \mathbf{X}((\neg q_1)\mathbf{W}(q_2 \wedge \neg q_1)))$.

**D14** defines an initial precedence connective and **D15** an iterated precedence connective. Both connectives are anchored; precedence is required only after the indexing formula occurs. In specifications, indexes are instantiated with **beg** and omitted. These connectives are needed to express causality. In our example, *get* and *reply* are causally connected, meaning that these events do not happen concurrently and each occurrence of *get* causes a subsequent dispatch of *reply*, which does not happen otherwise (11.11, 11.14). This shows that their occurrence is alternating. Note that neither of the connectives above is definable in terms of **X**, **F** and **G** only, justifying our choice of a temporal logic based on a strict strong until connective.

There exists just another actor primitive not treated so far: **become**, which prescribes that an actor will behave in its subsequent computation according to a distinct specification determined *a priori*. In fact, local computations in $\Gamma_c$ like *cons* (consumption) of our example together with a selective use of attribute symbols simulate this in an awkward manner. Indeed, the whole BufferCell specification could have been split so that each cell could become both a linked and an empty one according to the processing of previously received messages[3]. It would be easy to present **become** as another definition by introducing death actions in signatures and by defining the primitive as the death of an actor and its subsequent resurrection with a distinct behaviour, keeping the same mail address in this process. However, we have reasons to avoid treating this here: in the first place, in order to simplify our presentation, and, secondly, because the primitive, with the meaning described above, does not increase the expressive power of the model, as identified by Agha (1986).

Concerning the interpretation of signature symbols, the same assumptions made in the previous chapter are applicable here. Note in particular that events

[3]Note that, since $|\Gamma_{l_b}| \in [0, \omega_0[$, we allow actors to have “multiple constructors”.

may happen concurrently if this is allowed by specification axioms and thus action symbols are a syntactic representation of the events of Hewitt and Baker (1977), which may proceed concurrently if unrelated. Specifications are defined in terms of parameterised signatures in the usual way. Axioms are only satisfied by sets of infinite discrete sequences of worlds representing the behaviour of an actor community and this captures all the possible evolutions of an open system rather than just the possibly terminating behaviour of some particular objects.

## 3.2.2 Axiomatising Actor Behaviours

In this section, we develop a proof calculus for the actor model which particularises the logical system of the previous chapter by considering an additional set of logical axioms and inference rules. The associated notion of model is taken from the class of structures defined in Section 2.8 which also satisfy our extended axiomatisation. Thus, we can focus on the actor model here.

We develop an axiomatisation of a consequence relation $\vdash_\Delta$, which is indexed by a signature $\Delta$ because this relation is defined in a way that strictly depends on the symbols of the given signature. In other words, $\vdash$ is a weakly structural consequence relation. We assume that $\Delta = (\Sigma, \mathcal{A}, \Gamma)$ is given. We also use the variable $n$ for actor names, decorated with indexes whenever necessary. Moreover, for a given $c \in \Gamma$, $type(c) = \langle s_1, \ldots, s_n \rangle$, $n \in \vec{v_c}$ abbreviates $\bigvee\{n = v_{c_i} | type(v_{c_i}) = \mathsf{addr}; 1 \leq i \leq n\}$ and $\vec{v_c} = \vec{u_c}$ abbreviates $\bigwedge\{v_{c_i} = u_{c_i} | 1 \leq i \leq n\}$. Free variables in axioms are considered to be implicitly universally quantified and the following notation is used to express the invariance of an expression; that a required actor name has become known due to the delivery of a message, the birth of the actor or the creation of new objects; that a property does not occur until a specific actor name becomes known; and a strong fairness requirement over the occurrence of a particular formula:

| For | in | formula | represents |
|---|---|---|---|
| $t$ | $Term(\Delta)$ | $Inv(t)$ | $\forall k \cdot t = k \rightarrow \mathbf{X}(t = k)$ |
| $p$ | $\mathcal{G}(\Delta)$ | $Inv(p)$ | $(p \wedge \mathbf{X}p) \vee (\neg p \wedge \mathbf{X}(\neg p))$ |
| $n$ | $Term(\Delta)_{\mathsf{addr}}$ | $Acq(n)$ | $\bigvee\{\exists \vec{v_d} \cdot \mathbf{deliv}\ (d, \vec{v_d}) \wedge n \in \vec{v_d} \mid d \in \Gamma_{l-l_b}\}$ <br> $\bigvee\{\exists \vec{v_d} \cdot d(\vec{v_d}) \wedge n \in \vec{v_d} \mid d \in \Gamma_{l_b}\}$ <br> $\bigvee\{\exists \vec{v_d} \cdot \mathbf{new}(d, n, \vec{v_d}) \mid d \in \Gamma_{e_b}\}$ |
| $n, p$ | $Term(\Delta)_{\mathsf{addr}}, \mathcal{G}(\Delta)$ | $Wait(n, p)$ | $(\neg p)\mathbf{W}(\mathbf{init}) \wedge (\neg p)\mathbf{W}(Acq(n))$ |
| $p$ | $\mathcal{G}(\Delta)$ | $Fair(p)$ | $\mathbf{F}(p \vee \mathbf{GA}(\neg p))$ |

As identified by Hewitt and Baker (1977), *locality* is an essential characteristic of the actor model. This is also a crucial assumption in object-based logics to support modular specification and reasoning (Fiadeiro and Maibaum 1992, Sernadas *et al.* 1995). Generally speaking, locality requires that state changes of an actor be effected only by the events related to the object itself. This means in particular that each actor has encapsulated state. We choose to capture locality through the axioms below:

$(\overline{\textbf{L1}})\ \bigvee_{c\in\Gamma_c} \exists\vec{v_c}\cdot n.c(\vec{v_c}) \vee \bigwedge_{f\in\mathcal{A}_l} \forall\vec{v_f}\cdot n.Inv(f(\vec{v_f}))$

$(\textbf{L2})\ \bigwedge_{c\in\Gamma_{l_b}} \forall\vec{v_c}\cdot\exists n_1\cdot n_1.\textbf{new}(c,n_2,\vec{v_c}) \vee n_2.Inv(init_c(\vec{v_c}))$

$(\textbf{L3})\ \bigwedge_{c\in\Gamma_{l-l_b}} \forall\vec{v_c}\cdot\exists n_2\cdot n_2.\textbf{send}\ c,n_1,\vec{v_c}\ (\vee)n_1.\textbf{deliv}\ (c,\vec{v_c}) \vee n_1.Inv(sent_c(\vec{v_c}))$

$(\textbf{L4})\ \bigwedge_{c\in\Gamma_{l-l_b}} \forall\vec{v_c}\cdot n_1.\textbf{deliv}\ (c,\vec{v_c}) \vee n_1.c(\vec{v_c}) \vee n_1.Inv(delivd_c(\vec{v_c}))$

The first axiom says that either an actor performs a local computation or its extra-logical attributes all remain invariant. In the BUFFERCELL example, this means that either *cons*, *link* or *go* occur or else the values of *val*, *nxt*, *void*, *lst* and *up* do not change. According to the second axiom, either an object is created with a certain name or the existence of an actor with such a name is not disturbed. The other two logical axioms are to guarantee that buffering attributes vary only when message passing takes place.

The following axioms constrain the *occurrence* of events:

$(\overline{\textbf{O1}})\ \bigwedge_{c\in\Gamma_{e-e_b}} \forall\vec{v_c}\cdot\textbf{beg}\rightarrow\textbf{G}(\neg n_1.\textbf{init}) \vee \bigwedge_{n\in n_2:\vec{v_c}} n_1.Wait(n,\textbf{send}\ c,n_2,\vec{v_c}\ ())$

$(\overline{\textbf{O2}})\ \bigwedge_{c\in\Gamma_{l-l_b}} \forall\vec{v_c}\cdot\textbf{beg}\rightarrow(\neg n.\textbf{deliv}\ (c,\vec{v_c}))\textbf{W}(n.\textbf{init})$

$(\overline{\textbf{O3}})\ \bigwedge_{c\in\Gamma_{(l-l_b)\cup c}} \forall\vec{v_c}\cdot\textbf{beg}\rightarrow(\neg n.c(\vec{v_c}))\textbf{W}(n.\textbf{init})$

$(\overline{\textbf{O4}})\ \bigwedge_{c\in\Gamma_{e_b}} \forall\vec{v_c}\cdot\textbf{beg}\rightarrow\textbf{G}(\neg n_1.\textbf{init}) \vee \bigwedge_{n\in\vec{v_c}} n_1.Wait(n,\exists n_2\cdot\textbf{new}(c,n_2,\vec{v_c}))$

$(\textbf{O5a})\ \bigwedge_{\substack{b,c,\in\Gamma_{l_b}\\ d\in\Gamma l-l_c}} \forall\vec{v_c},\vec{v_d}\cdot\exists n_1,\vec{v_b}\cdot n_1.\textbf{new}(b,n_2,\vec{v_b})\rightarrow n_2.init_d(\vec{v_c})=n_2.sent_d(\vec{v_d})=n_2.delivd_d(\vec{v_d})=\mathrm{F}$

$(\textbf{O5b})\ \bigwedge_{c\in\Gamma_{l_b}} \forall\vec{v_c}\cdot\textbf{beg}\rightarrow(n.c(\vec{v_c})\leftrightarrow n.init_c(\vec{v_c})=\mathrm{T})$

$(\overline{\textbf{O6a}})\ \bigwedge_{c\in\Gamma_{l_b}} \exists n_1,n_2,\vec{v_c}\cdot\textbf{E}(n_1.\textbf{new}(c,n_2,\vec{v_c}))$

$(\overline{\textbf{O6b}})\ \bigwedge_{c\in\Gamma_{l_b}} \textbf{G}(\exists!\,n_2\cdot\exists n_1,\vec{v_c}\cdot n_1.\textbf{new}(c,n_2,\vec{v_c}))\rightarrow\forall n_2\cdot\textbf{F}(\exists n_1,\vec{v_c}\cdot n_1.\textbf{new}(c,n_2,\vec{v_c}))$

$(\overline{\textbf{O7a}})\ \bigwedge_{c\in\Gamma_{l_b}} \forall\vec{v_c}\cdot\exists n_1\cdot n_1.\textbf{new}(c,n_2,\vec{v_c})\rightarrow\textbf{XF}(n_2.c(\vec{v_c}))$

$(\overline{\textbf{O7b}})\ \bigwedge_{c\in\Gamma_{l_b}} \forall\vec{v_c}\cdot\textbf{beg}\rightarrow\textbf{X}((\neg n_2.c(\vec{v_c}))\textbf{W}(\neg n_2.c(\vec{v_c})\wedge\exists n_1\cdot n_1.\textbf{new}(c,n_2,\vec{v_c})))$

$(\overline{\textbf{O8}})\ \bigwedge_{\substack{c,d\in\Gamma_{l_b}\\ d\neq c}} \forall\vec{v_c}\cdot n_1.\textbf{new}(c,n_2,\vec{v_c})\rightarrow\nexists n_3,\vec{u_c},\vec{v_d}\cdot n_3:\vec{u_c}\neq n_1:\vec{v_c}\wedge n_3.\textbf{new}(c,n_2,\vec{u_c})\vee n_3.\textbf{new}(d,n_2,\vec{v_d})$

$(\textbf{O9})\ \bigwedge_{c\in\Gamma_{l-l_b}} \forall\vec{v_c}\cdot n.\textbf{deliv}\ (c,\vec{v_c})\rightarrow n.sent_c(\vec{v_c})=\mathrm{T}$

$(\overline{\textbf{O10}})\ \bigwedge_{\substack{c,d\in\Gamma_{l-l_b}\\ d\neq c}} \forall\vec{v_c}\cdot n.\textbf{deliv}\ (c,\vec{v_c})\rightarrow\nexists\vec{u_c},\vec{v_d}\cdot\vec{u_c}\neq\vec{v_c}\wedge n.\textbf{deliv}\ (c,\vec{u_c})\vee n.\textbf{deliv}\ (d,\vec{v_d})$

$$(\mathbf{O11})\ \bigwedge_{c \in \Gamma_{l - l_b}} \forall \vec{v_c} \cdot n.c(\vec{v_c}) \rightarrow n.delivd_c(\vec{v_c}) = \mathrm{T}$$

$$(\overline{\mathbf{O12}})\ \bigwedge_{\substack{c,d \in \Gamma_{(l - l_b) \cup c} \\ c \neq d}} \forall \vec{v_c} \cdot n.c(\vec{v_c}) \rightarrow \nexists \vec{u_c}, \vec{v_d} \cdot \vec{u_c} \neq \vec{v_c} \wedge n.c(\vec{u_c}) \vee n.d(\vec{v_d})$$

**O1-4** state that, before the birth of an actor, not only the dispatch, delivery and consumption of messages but also local computations and requests for creation are forbidden. Note that **O1** and **O4** are more liberal than the other axioms if the respective actor is never created but are more restrictive otherwise by requiring that each actor name becomes known due to the delivery of a message, the birth of the actor or the creation of another object before the name can be used in the respective task. These restrictions are to prevent the use of arbitrary names and modes of interaction such as broadcasting which are distinct from point-to-point message passing. On the other hand, the same axioms are permissive concerning unborn actors because we are capturing an open mode of interaction, which cannot be totally constrained by the local semantics. An actor complying with some community specification, say, does not have to be created in this context, but may need to dispatch some messages which are mentioned in the specification. Therefore, the occurrence of these events should not be logically forbidden. The situation above is dual to that described by Fiadeiro and Maibaum (1997) wherein read-only attributes are adopted as a means of capturing an open synchronous mode of interaction. Such attributes cannot be constrained locally, but only at a global level where the respective components are put together and interfere with the behaviour of one another.

The subsequent set of logical axioms above relates the creation of new actors, the occurrence of birth actions and the existence of other objects. **O5a** and the other axioms imply that an actor can only be created once and also that messages are not sent or delivered to the object before its birth. Moreover, according to **O5b**, the actor birth occurs in the beginning of time if the object always exists. **O6a** says that it is always possible for some actor to create a new object and **O6b** states that all the actor names will be used if exactly one object is created at each instant. It is important to mention that, because of the specific characteristics of the adopted time flows, the former axiom implies that the set of actor names is infinite while the latter implies that the same set is countable. **O7a** and **O7b** state that the occurrence of births and requests for creation are always causally connected after the initial moment.

We have also proposed a set of axioms stating mutual exclusion. Most of these properties are particular to the actor model, whereas a few are due to decisions in the design of our formalism. **O8** specifies that actors with the same name cannot be concurrently created; **O9** says that messages can be delivered only if they were previously sent; **O10** determines that only one message can be delivered to an actor at each instant; **O11** says that messages can be consumed only if they were previously delivered; and finally, according to **O12**, message consumption and local computations of an actor are totally ordered, meaning that two such events cannot occur in parallel. Concerning this last axiom, we could have allowed instead actors with full internal concurrency while ensuring attribute consistency through additional axioms. We prefer the simpler formulation here to facilitate specification and reasoning. Note that the specified actors can always present some internal concurrency anyway: they can, for instance, create many other objects and send several messages at the same time.

Many logical attributes are introduced in the extension of actor signatures. The modification of their values according to the occurrence of the respective actions is defined by the following *valuation* axioms:

$$(\mathbf{V1})\ \bigwedge_{c\in\Gamma_{l_b}} \forall\vec{v_c}\cdot\exists n_1\cdot n_1.\mathbf{new}(c,n_2,\vec{v_c})\to\mathbf{X}(n_2.init_c(\vec{v_c})=\mathrm{T})$$

$$(\mathbf{V2})\ \bigwedge_{c\in\Gamma_{l-l_b}} \forall\vec{v_c}\cdot\exists n_1\cdot n_1.\mathbf{send}\ c,n_2,\vec{v_c}\ (\to)\mathbf{X}(n_2.sent_c(\vec{v_c})=\mathrm{T})$$

$$(\mathbf{V3})\ \bigwedge_{c\in\Gamma_{l-l_b}} \forall\vec{v_c}\cdot n.\mathbf{deliv}\ (c,\vec{v_c})\to\mathbf{X}(n.sent_c(\vec{v_c})=\mathrm{F}\wedge n.delivd_c(\vec{v_c})=\mathrm{T})$$

$$(\mathbf{V4})\ \bigwedge_{c\in\Gamma_{l-l_b}} \forall\vec{v_c}\cdot n.c(\vec{v_c})\to\mathbf{X}(n.delivd_c(\vec{v_c})=\mathrm{F})$$

According to **V1**, if the creation of an actor has been requested, there will exist a new actor in the next instant. Moreover, axioms **V2** and **V3** say that if a message is dispatched, it will be buffered for output, and likewise the message will be removed from the output and transferred to the input buffer whenever it is delivered. Furthermore, each processed message will be subsequently removed from the input buffer as stated in axiom **V4**. Note that the delay in buffering messages, in the next instant only, rules out the existence of Zeno actors, which could receive, compute and reply infinitely fast.

Finally, *fairness* axioms are required to guarantee a correct collective behaviour. Without fairness, it could be the case that a message is not delivered even if the target actor is always willing to receive it, e.g., because of a transmission failure, and likewise that received messages are never consumed.

$$(\mathbf{F1})\ \bigwedge_{c\in\Gamma_{l-l_b}} \forall\vec{v_c}\cdot n.delivd_c(\vec{v_c})=\mathrm{T}\wedge\mathbf{E}(n.c(\vec{v_c}))\to n.Fair(c(\vec{v_c}))$$

$$(\mathbf{F2})\ \bigwedge_{c\in\Gamma_{l-l_b}} \forall\vec{v_c}\cdot n.sent_c(\vec{v_c})=\mathrm{T}\wedge\mathbf{E}(n.\mathbf{deliv}\ (c,\vec{v_c}))\to n.Fair(\mathbf{deliv}\ (c,\vec{v_c}))$$

The first axiom says that, if the processing of a single message is obliged, because the message was delivered and has been locally buffered, and it is also enabled, i.e., possible, the message will be processed or else the actor will become always disabled for processing, unable to consume the pending message. *Mutatis* *mutandis*, this is what the second axiom says for message delivery. These axioms capture assumptions that can be classified in between those of perfect and initially perfect buffers as described by Koymans (1987).

A crucial simplification has been made here concerning message passing. We should have treated the fact that messages may be exchanged in sequence or concurrently and some of them could be lost or duplicated in this way. The usual treatment of this problem is to attach tags to messages so that they become distinct from each other. To avoid obliging the specifier to deal with such details, a logical treatment could have been defined here, much in the way that object naming is dealt with through auxiliary attributes. Details are omitted.

All the properties discussed above have already been stated in the literature on the actor model, e.g. by Clinger (1981), Hewitt and Baker (1977), despite the lack of a formally stated axiomatisation. Hereafter, we name the full set of logical axioms as $Ax \overset{\text{def}}{=} \{$**L1-4**, **O1-12**, **V1-4**, **F1-2**$\}$. The set $\overline{Ax}$, on the other hand, contains only the axioms with barred labels, wherein logical attribute symbols do not appear. The axiomatisation of the actor model allows us to derive the following more or less standard temporal logical rules for reasoning about the concurrent behaviour of object communities:

**Proposition 3.2.4 (Derived Rules of Inference)** Given an actor specification $\Phi = (\Delta, \Psi)$, $\Delta = (\Sigma, \mathcal{A}, \Gamma)$, the following inference rules are derivable for existing objects in the $\Phi$ community, provided that $\{k, n_1, n_2\} \subset \mathcal{V}^{MSBTL}_{\mathsf{addr}}$, $p_1$ and $q$ are local state formulas parameterised by $n_1$ and $p_2$ is a local state formula parameterised by $n_2$:

$$\boxed{\begin{array}{rl}
(\textbf{EXIST}) & \begin{array}{ll} 1. & p_2[k] \rightarrow \exists \vec{v_b} \cdot n_2.\mathbf{new}(b, k, \vec{v_b}) \\ 2. & p_1[k] \rightarrow q \vee \bigvee_{c \in \Gamma_{l_b}} \exists \vec{v_c} \cdot n_1.\mathbf{new}(c, k, \vec{v_c}) \end{array} \\
b \in \Gamma_{e_b} & \overline{\hphantom{p_1[k] \rightarrow q \vee \bigvee_{c \in \Gamma_{l_b}} \exists \vec{v_c} \cdot n_1.\mathbf{new}(c, k, \vec{v_c})}} \\
& p_2[k] \rightarrow \mathbf{XG}(p_1[k] \rightarrow q)
\end{array}}$$

$$\boxed{\begin{array}{rl}
(\textbf{SAFE}) & \begin{array}{ll} 1. & \bigwedge_{b \in \Gamma_{l_b}} \forall \vec{v_b} \cdot n_1.b(\vec{v_b}) \rightarrow q \\ 2. & \bigwedge_{c \in \Gamma_c} \forall \vec{v_c} \cdot n_1.c(\vec{v_c}) \wedge q \rightarrow \mathbf{X}q \end{array} \\
\hline
& \mathbf{G}q
\end{array}}
\qquad
\boxed{\begin{array}{rl}
(\textbf{INV}) & 1. \quad \bigwedge_{c \in \Gamma_c} \forall \vec{v_c} \cdot n_1.c(\vec{v_c}) \wedge q \rightarrow \mathbf{X}q \\
\hline
& q \rightarrow \mathbf{G}q
\end{array}}$$

$$\boxed{\begin{array}{rl}
(\textbf{RESP}) & \begin{array}{ll} 1. & \bigwedge_{c \in \Gamma_c} \forall \vec{v_c} \cdot n_1.c(\vec{v_c}) \wedge p_1[\vec{v_d}] \rightarrow \mathbf{X}(p_1[\vec{v_d}] \vee n_1.d(\vec{v_d})) \\ 2. & n_1.d(\vec{v_d}) \rightarrow \mathbf{F}(q[\vec{v_d}]) \\ 3. & p_1[\vec{v_d}] \rightarrow \mathbf{FE}(n_1.d(\vec{v_d})) \end{array} \\
d \in \Gamma_{l-l_b} & \overline{\hphantom{\bigwedge_{c \in \Gamma_c} \forall \vec{v_c} \cdot n_1.c(\vec{v_c}) \wedge p_1[\vec{v_d}] \rightarrow \mathbf{X}(p_1[\vec{v_d}] \vee n_1.d(\vec{v_d}))}} \\
& n_1.\mathbf{deliv}\ (d, \vec{v_d}) \rightarrow \mathbf{X}(\mathbf{F}(p_1[\vec{v_d}]) \rightarrow \mathbf{F}(q[\vec{v_d}]))
\end{array}}$$

$$\boxed{\begin{array}{rc} (\textbf{COM}) & \\ d \in \Gamma_{e-e_b}\text{ and} & \\ d \in \Gamma_{l-l_b} & \end{array} \dfrac{\begin{array}{ll} 1. & \bigwedge_{c\in\Gamma_c} \forall \vec{v_c} \cdot n_1.c(\vec{v_c}) \wedge p_1[\vec{v_d}] \rightarrow \mathbf{X}(p_1[\vec{v_d}] \vee n_1.\mathbf{deliv}\ (d, \vec{v_d})) \\ 2. & n_1.\mathbf{deliv}\ (d, \vec{v_d}) \rightarrow \mathbf{F}(q[\vec{v_d}]) \\ 3. & p_1[\vec{v_d}] \rightarrow \mathbf{FE}(n_1.\mathbf{deliv}\ (d, \vec{v_d})) \end{array}}{n_2.\mathbf{send}\ d, n_1, \vec{v_d}\ (\rightarrow)\mathbf{X}(\mathbf{F}(p_1[\vec{v_d}]) \rightarrow \mathbf{F}(q[\vec{v_d}]))}}$$

$$\boxed{\begin{array}{rc} (\textbf{NRESP}) & \\ b \in \Gamma_{l_b \cap e_b} & \\ d \in \Gamma_{l-l_b} & \end{array} \dfrac{\begin{array}{ll} 1. & p_2[n_1] \rightarrow \exists \vec{v_b} \cdot n_2.\mathbf{new}(b, n_1, \vec{v_b}) \\ 2. & \exists \vec{v_b} \cdot n_1.b(\vec{v_b}) \rightarrow \forall v_d \cdot p_1[\vec{v_d}] \\ 3. & n_1.d(\vec{v_d}) \rightarrow q[\vec{v_d}] \\ 4. & \exists n \cdot n.\mathbf{deliv}\ (d, \vec{v_d}) \rightarrow \mathbf{X}(q[\vec{v_d}]) \\ 5. & n_1.d(\vec{v_d}) \rightarrow \mathbf{X}(p_1[\vec{v_d}] \vee q[\vec{v_d}]) \end{array}}{p_2[n_1] \rightarrow \mathbf{XG}(p_1[\vec{v_d}] \rightarrow (\neg n_1.d(\vec{v_d}))\mathbf{W}(n_1.\mathbf{deliv}\ (d, \vec{v_d})))}}$$

$$\boxed{\begin{array}{rc} (\textbf{NCOM}) & \\ b \in \Gamma_{l_b \cap e_b} & \\ d \in \Gamma_{e-e_b}\text{ and} & \\ d \in \Gamma_{l-l_b} & \end{array} \dfrac{\begin{array}{l} 1.\ p_2[n_1] \rightarrow \exists \vec{v_b} \cdot n_2.\mathbf{new}(b, n_1, \vec{v_b}) \\ 2.\ \exists \vec{v_b} \cdot n_1.b(\vec{v_b}) \rightarrow \forall v_d \cdot p_1[\vec{v_d}] \\ 3.\ n_1.\mathbf{deliv}\ (d, \vec{v_d}) \rightarrow q[\vec{v_d}] \\ 4.\ \exists n \cdot n.\mathbf{send}\ d, n_1, \vec{v_d}\ (\rightarrow)\mathbf{X}(q[\vec{v_d}]) \\ 5.\ n_1.\mathbf{deliv}\ (d, \vec{v_d}) \rightarrow \mathbf{X}(p_1[\vec{v_d}] \vee q[\vec{v_d}]) \end{array}}{p_2[n_1] \rightarrow \mathbf{XG}(p_1[\vec{v_d}] \rightarrow (\neg n_1.\mathbf{deliv}\ (d, \vec{v_d}))\mathbf{W}(\exists n \cdot n.\mathbf{send}\ d, n_1, \vec{v_d}\ ()))}}$$

The rules above can be derived using the axiomatisation of the branching time logic and our logical axioms about the actor model. These rules are more convenient to use because the logical attributes have been eliminated. Rule **EXIST**, based on the fact that a name cannot be reused once it is given to some actor, guarantees a local safety property from the configuration of the actors in the environment. **SAFE** and **INV** are the usual rules for verifying safety and invariance properties. Rules **COM** and **RESP** capture the fairness requirements on actor behaviours. They should be applied to verify that the consequences of delivering or consuming a message are eventually obtained whenever the recipient actor becomes enabled often enough to guarantee the occurrence of the respective event. The slightly more complex rules for absence of communication and response, **NCOM** and **NRESP**, respectively, need to be ground on the creation of new actors since our axiomatisation admits initially present messages addressed to originally existing objects. Their conclusions are that, once the actor is created, whenever there are no pending messages for delivery or processing, messages will be delivered or consumed only if preceded by the occurrence of their triggering events. All these inference rules may be simplified by a careful instantiation of the adopted schematic variables.

The rule **COM** in particular is to be used in proving properties that arise from the interaction between two (potentially distinct) actors. The situation here differs from that described in (Barreiro *et al.* 1995), where interaction is captured via action sharing in a more explicit and unconstrained manner. Therein, a very strong form of fairness is proposed, since in general a shared action may loose permission to happen in some component while obliged to take place. Considering actor systems, however, such a fairness strengthening is not required: an event must be locally provided by one actor only and cannot have its permission to occur externally constrained in this way.

## 3.3 Verification of Local Properties

Let us illustrate the application of our specific proof calculus to the verification of local properties of individual actors. From the BufferCell specification, it is easy to see that once a cell is created, it may be consumed and linked to another cell of the buffer afterwards. If a cell has already been consumed and it is not the last element of the list, the cell will never perform such local computations again. Hence, the cell will simply forward every incoming message to the subsequent buffer element. The previous property is stated as follows:

$$\vdash_{\textsc{BufferCell}} \mathit{void} = \mathrm{T} \wedge \mathit{lst} = \mathrm{F} \rightarrow \mathbf{G}(\neg \mathit{cons} \wedge \neg \mathit{link}(n)) \tag{3.3.1}$$

As in the examples of the previous chapter, we split the verification of this property into two parts, which are both developed in a similar way. We first deal with the action *cons*. The axioms in the specification are helpful in showing that *void*, part of the antecedent of the implication above, is always invariant after becoming true. To begin with the proof, let us examine the effect of the action *go* over the value of this attribute:

| | | |
|---|---|---|
| 1. | $go \wedge val = v \wedge void = x \wedge nxt = n \wedge lst = y \rightarrow \mathbf{X}(val = v \wedge void = x \wedge nxt = n \wedge lst = y)$ | (11.5) |
| 2. | $go \wedge val = v \wedge void = x \wedge nxt = n \wedge lst = y \rightarrow \mathbf{X}(void = x)$ | **DIST-ANDX**, **HS** 1 **DIST-IFA**, **R1-MP**, **AND-E** + |
| 3. | $val = v \rightarrow (go \wedge void = x \wedge nxt = n \wedge lst = y \rightarrow go \wedge val = v \wedge void = x \wedge nxt = n \wedge lst = y)$ | **A1-I**, **A1-I**, **REFL**, **R1-MP** **AND-R**, **DIST-IFA**, **HS** + |
| 4. | $val = v \rightarrow (go \wedge void = x \wedge nxt = n \wedge lst = y \rightarrow \mathbf{X}(void = x))$ | **LTRAN**, **R1-MP** 2, **LTRAN** **R1-MP**, **R1-MP** 3 + |
| 5. | $go \wedge void = x \wedge nxt = n \wedge lst = y \rightarrow \mathbf{X}(void = x)$ | **GEN-∀** 4, **EXC-∀∃**, **R1-MP** **NVOID**, **R1-MP** + |
| 6. | $nxt = n \rightarrow (go \wedge void = x \wedge lst = y \rightarrow go \wedge void = x \wedge nxt = n \wedge lst = y)$ | **A1-I**, **A1-I**, **REFL**, **R1-MP** **AND-R**, **DIST-IFA**, **HS** + |
| 7. | $nxt = n \rightarrow (go \wedge void = x \wedge lst = y \rightarrow \mathbf{X}(void = x))$ | **LTRAN**, **R1-MP** 5, **LTRAN** **R1-MP**, **R1-MP** 6 + |
| 8. | $go \wedge void = x \wedge lst = y \rightarrow \mathbf{X}(void = x)$ | **GEN-∀** 7, **EXC-∀∃**, **R1-MP** **NVOID**, **R1-MP** + |

| | | |
|---|---|---|
| 9. | $lst = y \rightarrow (go \wedge void = x \rightarrow$ | **A1-I**, **A1-I**, **REFL**, **R1-MP** |
| | $go \wedge void = x \wedge lst = y)$ | **AND-R**, **DIST-IFA**, **HS** + |
| 10. | $lst = y \rightarrow$ | **LTRAN**, **R1-MP** 8, **LTRAN** |
| | $(go \wedge void = x \rightarrow \mathbf{X}(void = x))$ | **R1-MP**, **R1-MP** 9 + |
| 11. | $go \wedge void = x \rightarrow \mathbf{X}(void = x)$ | **GEN-**$\forall$ 10, **EXC-**$\forall\exists$, **R1-MP**, **NVOID**, **R1-MP** |
| 12. | $go \wedge void = \mathrm{T} \rightarrow \mathbf{X}(void = \mathrm{T})$ | **GEN-**$\forall$ 11, **A19-**$\forall$, **R1-MP** |

The rationale behind the verification of the following two sentences is the same as adopted above. Hence, the respective derivations can safely be omitted. Then, we are allowed to conjoin these and sentence (12) above in order to obtain the required premise for an application of rule **INV**, completing in this way the verification that *void* is always invariant after becoming true.

| | | |
|---|---|---|
| 13. | $link(n) \wedge void = \mathrm{T} \rightarrow \mathbf{X}(void = \mathrm{T})$ | from 11.8 |
| 14. | $cons \rightarrow \mathbf{X}(void = \mathrm{T})$ | from 11.6 |
| 15. | $cons \wedge void = \mathrm{T} \rightarrow \mathbf{X}(void = \mathrm{T})$ | **AND-L** 14 |
| 16. | $void = \mathrm{T} \rightarrow \mathbf{G}(void = \mathrm{T})$ | **AND-I** 12, 13; **AND-I** 15; **INV** |

Now we have to ensure that a buffer cell cannot be consumed more than once. The following implication can be used to simplify considerably the specification axiom involved:

$$(\neg(cons \vee \mathbf{send}\ reply, n, v\ ()))\mathbf{W}(get(n) \wedge val = v \wedge void = \mathrm{F} \wedge \neg cons) \rightarrow (\neg cons)\mathbf{W}(get(n) \wedge void = \mathrm{F} \wedge \neg cons) \tag{3.3.2}$$

This sentence is provable based on **MON-GW**, which captures the monotonicity of the connective **W**. The main derivation proceeds as follows:

| | | |
|---|---|---|
| 17. | $cons \rightarrow$ | (11.14), **D15-P**, **AND-E**, (3.3.2), |
| | $\mathbf{X}((\neg cons)\mathbf{W}(get(n) \wedge void = \mathrm{F} \wedge \neg cons))$ | **R2-G**, **MON-GX**, **R1-MP**, **HS** + |
| 18. | $(\neg cons)\mathbf{W}(get(n) \wedge void = \mathrm{F} \wedge \neg cons) \rightarrow$ | **REFL**, **D10-W**, **RPL-UF**, **OR-R** |
| | $\mathbf{F}(get(n) \wedge void = \mathrm{F} \wedge \neg cons) \vee \mathbf{G}(\neg cons)$ | **REFL**, **OR-R**, **OR-L**, **HS** + |
| 19. | $\mathbf{F}(get(n) \wedge void = \mathrm{F} \wedge \neg cons) \rightarrow$ | $\mathsf{bool}\ Ax$, **AND-L**, **R2-G** |
| | $\mathbf{F}(\neg void = \mathrm{T})$ | **MON-GF**, **R1-MP** + |
| 20. | $(\neg cons)\mathbf{W}(get(n) \wedge void = \mathrm{F} \wedge \neg cons) \rightarrow$ | **INVE**, **R1-MP** 19, **RTRAN**, **R1-MP** |
| | $(\mathbf{G}(void = \mathrm{T}) \rightarrow \mathbf{G}(\neg cons))$ | **R1-MP**, **D3-OR**, **D9-G**, **HS** 18 + |
| 21. | $\mathbf{X}((\neg cons)\mathbf{W}(get(n) \wedge void = \mathrm{F} \wedge \neg cons)) \rightarrow$ | **R2-G** 20, **MON-GX**, **R1-MP** |
| | $(\mathbf{XG}(void = \mathrm{T}) \rightarrow \mathbf{XG}(\neg cons))$ | **MON-GX**, **HS** + |
| 22. | $cons \rightarrow \mathbf{XG}(void = \mathrm{T})$ | **R2-G** 16, **MON-GX**, **R1-MP**, **HS** 14 |
| 23. | $cons \rightarrow \mathbf{XG}(\neg cons)$ | **HS** 17, 21; **PERM**, **R1-MP** |
| | | **HS** 22, **CONT**, **R1-MP** + |

The action *cons* does not happen spontaneously. Indeed, it is caused by the consumption of a message *get*. Based on the logical axiom **O12**, which forbids the occurrence of local computations in parallel with message consumptions, and **L1**, which forces each actor to have an encapsulated state, we take advantage of this causality relation to relate the actor state and the occurrence of local computations:

| | | |
|---|---|---|
| 24. | $get(n) \wedge val = v \wedge void = \mathrm{F} \rightarrow$ | **DIST-ANDX**, (11.11), **HS** |
| | $\mathbf{X}(cons)$ | **DIST-IFA**, **AND-E** + |
| 25. | $val = v \rightarrow (get(n) \wedge void = \mathrm{F} \rightarrow$ | **A1-I**, **A1-I**, **REFL**, **R1-MP** |
| | $get(n) \wedge val = v \wedge void = \mathrm{F})$ | **AND-R**, **DIST-IFA**, **HS** + |
| 26. | $val = v \rightarrow$ | **LTRAN**, **R1-MP** 24, **LTRAN** |
| | $(get(n) \wedge void = \mathrm{F} \rightarrow \mathbf{X}(cons))$ | **LTRAN**, **R1-MP**, **R1-MP** 25 + |
| 27. | $\exists v \cdot val = v \rightarrow (get(n) \wedge void = \mathrm{F} \rightarrow \mathbf{X}(cons))$ | **GEN-**$\forall$ 26, **EXC-**$\forall\exists$, **R1-MP** |
| 28. | $get(n) \wedge void = \mathrm{F} \rightarrow \mathbf{X}(cons)$ | **NVOID**, **R1-MP** 27 |
| 29. | $\mathbf{G}(\neg cons) \rightarrow \mathbf{G}(void = \mathrm{T} \rightarrow \neg cons)$ | **A1-I**, **R2-G**, **MON-G**, **R1-MP** |
| 30. | $\mathbf{XG}(\neg cons) \rightarrow \mathbf{XG}(void = \mathrm{T} \rightarrow \neg cons)$ | **R2-G** 29, **MON-GX**, **R1-MP** |
| 31. | $\mathbf{G}(cons \rightarrow \mathbf{XG}(void = \mathrm{T} \rightarrow \neg cons))$ | **HS** 23, 30; **R2-G** |
| 32. | $get(n) \wedge void = \mathrm{F} \rightarrow$ | **MON-GX** |
| | $\mathbf{XXG}(void = \mathrm{T} \rightarrow \neg cons)$ | **R1-MP** 31, **HS** 28 + |
| 33. | $get(n) \wedge void = \mathrm{F} \rightarrow$ | **O12**, **A19-**$\forall$ |
| | $\neg cons \wedge \neg link(k) \wedge \neg go$ | **HS**, **AND-L** + |
| 34. | $get(n) \wedge void = \mathrm{F} \rightarrow$ | **DIST-ANDX**, **DIST-IF**, **HS**, **DM**, **HS** 33 |
| | $\mathbf{X}(void = \mathrm{F})$ | **L1**, **D3-OR**, **HS**, **R1-MP**, **AND-E** + |
| 35. | $get(n) \wedge void = \mathrm{F} \rightarrow \mathbf{X}(void = \mathrm{F} \wedge cons)$ | **AND-R** 28, 34, **DIST-IFA**, **HS** |
| 36. | $(cons \rightarrow void = \mathrm{F}) \rightarrow (cons \rightarrow \neg void = \mathrm{T})$ | bool $Ax$, **LTRAN**, **R1-MP** |
| 37. | $void = \mathrm{F} \wedge cons \rightarrow (void = \mathrm{T} \rightarrow \neg cons)$ | **A1-I**, **AND-E**, **HS** 36, **INVE**, **HS** |
| 38. | $get(n) \wedge void = \mathrm{F} \rightarrow$ | **R2-G** 37, **MON-GX** |
| | $\mathbf{X}(void = \mathrm{T} \rightarrow \neg cons)$ | **R1-MP**, **HS** 35 + |
| 39. | $get(n) \wedge void = \mathrm{F} \rightarrow$ | **AND-R** 32, 38; **FIX-G**, **R2-G** |
| | $\mathbf{XG}(void = \mathrm{T} \rightarrow \neg cons)$ | **MON-GX**, **R1-MP**, **HS** + |
| 40. | $get(n) \wedge void = \mathrm{F} \rightarrow \neg cons$ | **DIST-IFA**, **HS** 33, **AND-E** |
| 41. | $get(n) \wedge void = \mathrm{F} \rightarrow (void = \mathrm{T} \rightarrow \neg cons)$ | **A1-I**, **HS** 40 |
| 42. | $get(n) \wedge void = \mathrm{F} \rightarrow \mathbf{G}(void = \mathrm{T} \rightarrow \neg cons)$ | **AND-R** 39, 41; **FIX-G**, **HS** |

We wish to conclude the proof that cells in the void state can never be consumed again. Note that this is true from the initial instant onwards. So, we can develop the remainder of the proof based on the specification axiom (11.14), which requires that since the beginning of time no *get* message be processed before the actor birth:

| | | |
|---|---|---|
| 43. | $\mathbf{beg} \rightarrow$ | (11.14), **D15-P** |
| | $(\neg cons)\mathbf{W}(get(n) \wedge void = \mathrm{F} \wedge \neg cons)$ | **AND-E**, (3.3.2), **HS** + |
| 44. | $\mathbf{beg} \rightarrow$ | **OR-L** 42, **D10-W** |
| | $\mathbf{G}(void = \mathrm{T} \rightarrow \neg cons) \vee (void = \mathrm{T} \rightarrow \neg cons)\mathbf{U}(\neg\top)$ | **TRAN-W** 43, **D10-W** + |
| 45. | $\neg\mathbf{F}(\neg\top) \rightarrow$ | **D3-OR** 44, **PERM**, **R1-MP** |
| | $(\mathbf{beg} \rightarrow \mathbf{G}(void = \mathrm{T} \rightarrow \neg cons))$ | **RPL-UF**, **INVE**, **R1-MP**, **HS** + |
| 46. | $\mathbf{beg} \rightarrow \mathbf{G}(void = \mathrm{T} \rightarrow \neg cons)$ | **D9-G** 45, **G**$\top$, **R1-MP** |

| | |
|---|---|
| 47. $\mathit{void} = \mathrm{T} \rightarrow \neg \mathit{cons}$ | **R3-begG** 46 |
| 48. $\mathbf{G}(\mathit{void} = \mathrm{T}) \rightarrow \mathbf{G}(\neg \mathit{cons})$ | **R2-G** 47, **MON-G**, **R1-MP** |
| 49. $\mathit{void} = \mathrm{T} \rightarrow \mathbf{G}(\neg \mathit{cons})$ | **HS** 16, 48 |

In a similar way, it can be shown that $\vdash_{\text{BufferCell}} \mathit{lst} = \mathrm{F} \rightarrow \mathbf{G}(\neg \mathit{link}(n))$. Conjoining these partial results based on **AND-L** and **DIST-IFA**, we conclude that (3.3.1) is derivable using **DIST-ANDG**. ■

The example above serves to illustrate important peculiarities in the verification of local safety properties of actors. Essentially, the same principles proposed in the previous chapter can be used to start this process. Note for instance that, due to the locality and local sequentiality assumptions, we could verify a buffer cell invariant using a case analysis argument based on the effect of each local computation over the attributes. However, because the occurrence of such local events is normally determined by the consumption of messages, we also have to rely on causality axioms to link both kinds of occurrence.

It is also interesting to note that, because we propose a set of logical axioms which takes into account the existence of a community of actors, here we have to particularise some of these axioms by removing the name of the respective actor from each expression in order to verify local safety properties. Because no interaction is involved, we may use just the set of axioms in $\overline{Ax}$ together with **SAFE** and **INV**, since logical symbols are introduced and axiomatised by the remaining logical axioms and rules precisely to support interaction. This whole process seems to be in contrast to the work of America and de Boer (1996), who adopt a three level axiomatisation and lift local sentences to intermediate and global contexts whenever necessary.

## 3.4 Composition of Actor Specifications

In Section 3.2.1 we discovered that, to give an account of what is usually considered to be a complex component in the actor model, we need at least to be able to put distinct signatures together to represent the linguistic structure of yet another component or an entire system. More generally, the view that complex descriptions should be defined in terms of simpler descriptions put together has been developed within the theory of Institutions by Goguen and Burstall (1992) and requires the definition of basic entities to be regarded as design units. In our case, they will be actor specifications.

It is also necessary to provide means of connecting object descriptions to each other. Traditionally, in a proof-theoretic approach to design, this is achieved by providing translations between the languages of the related theories (Maibaum and Turski 1984). If a symbol-to-symbol mapping, i.e., a morphism, between two actor signatures is given, the existence of a compositional relation of translation between the respective languages can be guaranteed.

**Definition 3.4.1 (Actor Signature Morphism)** Given two actor signatures $\Delta_1 = (\Sigma_1, \mathcal{A}_1, \Gamma_1)$ and $\Delta_2 = (\Sigma_2, \mathcal{A}_2, \Gamma_2)$, an *actor signature morphism* $\tau : \Delta_1 \to \Delta_2$ consists of:

- a morphism of algebraic structures $\tau_\upsilon : \Sigma_1 \to \Sigma_2$ such that $\tau_\upsilon(\mathsf{addr}_1) = \mathsf{addr}_2$;
- for each $f \in \mathcal{A}_{1_{\langle s_1,\ldots,s_n\rangle,s}}$, an attribute symbol $\tau_\alpha(f) : \tau_\upsilon(s_1)\times\ldots\times\tau_\upsilon(s_n) \to \tau_\upsilon(s)$ in $\mathcal{A}_2$;
- for each $c \in \Gamma_{1_{\langle s_1,\ldots,s_n\rangle}}$, an action symbol $\tau_\gamma(c) : \tau_\upsilon(s_1) \times \ldots \times \tau_\upsilon(s_n)$ in $\Gamma_2$ such that: (i) $\tau_\gamma(\Gamma_{c_1}) \subseteq \Gamma_{c_2}$; (ii) $\tau_\gamma(\Gamma_{e_{b_1}}) \subseteq \Gamma_{e_{b_2}}$; (iii) $\tau_\gamma(\Gamma_{e_1-e_{b_1}}) \subseteq \Gamma_{e_2-e_{b_2}}$; (iv) $\tau_\gamma(\Gamma_{l_{b_1}}) \subseteq \Gamma_{l_{b_2}}$; (v) $\tau_\gamma(\Gamma_{l_1-l_{b_1}}) \subseteq \Gamma_{l_2-l_{b_2}}$ and (vi) $\tau_\gamma(\Gamma_{l_1-e_1}) \subseteq \Gamma_{l_2-e_2}$.

It is straightforward to provide a compositional definition for the translation of classifications, terms, formulae and sets thereof under $\tau$. $\square$

Translations that necessarily relate the distinguished symbol of each signature, as defined above concerning $\mathsf{addr}$, have been called pointed morphisms in the literature (Parisi-Presicce and Pierantonio 1994). Since renaming is possible in translating the other signature symbols, morphisms capture the relabelling operation proposed by Agha (1986) to equalise identifiers in distinct descriptions. In addition, it is possible to use signature morphisms to allow some external symbols, members of $\Gamma_e$, to become local as well. This stems from the fact that, in a complex configuration, there may be events required from the environment of a component which are not provided by the environment of the whole configuration, because they are ensured by another component of the same configuration. It is not difficult to see that any given actor signature morphism induces other morphisms between the corresponding extended and parameterised signatures, by translating their additional symbols according to the way the original symbols are translated by the given morphism. This means that the specifier, in defining a morphism to connect two signatures, does not need to be concerned with the new symbols introduced in their extension or parameterisation.

We would like to be always able to combine any finite number of actor signatures so as to ensure the necessary structure to support interaction. This can be accomplished if we can show that actor signatures and morphisms determine a finitely co-complete category, as explained in the previous chapter:

**Theorem 3.4.2 (Category of Actor Signatures)** Actor signatures and morphisms constitute a finitely co-complete category $\mathbf{Sig}^{\mathbf{Act}}$.

Proof: To ensure that we have a category, we must show that identities exist and composition is associative. Considering that morphisms are set-valuated functions, the only difficulty that may arise in verifying the existence of identity is due to the non-disjoint sets of action symbols. But, for $\Delta \xrightarrow{\mathbf{id}} \Delta$, if $c \in \Gamma_{e_b}$, (a) $\mathbf{id}(c) \in \Gamma_{e_b}$, from (ii) and (iii) in the definition of signature morphisms. Now, if it is also the case that $c \in \Gamma_{l_b}$, (b) $\mathbf{id}(c) \in \Gamma_{l_b}$, from (iv) and (v). Due to (a) and (b), $\mathbf{id}(c) \in \Gamma_{e_b \cap l_b}$ whenever $c \in \Gamma_{e_b \cap l_b}$. The same argument applies to any $c \in \Gamma_{(e-e_b)\cap(l-l_b)}$ and therefore $\mathbf{Sig}^{\mathbf{Act}}$ admits identity, the constant function on sets. The associativity of signature morphisms follows directly from their set-theoretic definition.

The initial element of this category is $\Delta_\perp = ((\{\mathsf{addr}\}, \{\,\}), \{\,\}, \{\,\})$. Given a pair of morphisms $\Delta \xrightarrow{\tau_1} \Delta_1$, $\Delta \xrightarrow{\tau_2} \Delta_2$, their pushout is defined up to isomorphism by any pair of morphisms $\Delta_1 \xrightarrow{\tau_{1'}} \Delta'$, $\Delta_2 \xrightarrow{\tau_{2'}} \Delta'$ such that $S' = \tau_{1'}(S_1) \oplus_{\overline{\tau}(S)} \tau_{2'}(S_2)$, $\Omega' = \tau_{1'}(\Omega_1) \oplus_{\overline{\tau}(\Omega)} \tau_{2'}(\Omega_2)$, $\mathcal{A}' = \tau_{1'}(\mathcal{A}_1) \oplus_{\overline{\tau}(\mathcal{A})} \tau_{2'}(\mathcal{A}_2)$ and $\Gamma' = \tau_{1'}(\Gamma_1) \oplus_{\overline{\tau}(\Gamma)} \tau_{2'}(\Gamma_2)$, where $\overline{\tau} = \tau_1' \circ \tau_1 = \tau_2' \circ \tau_2$. The existence of the initial element and pushouts is sufficient to guarantee finite co-completeness. ■ **($\mathbf{Sig}^{\mathbf{Act}}$ Category)**

Specification morphisms induced by the signature morphisms above do not capture the expected enrichment of object behaviour as usual in Institutions (Goguen and Burstall 1992). This happens because they do not translate our additional logical axioms, which are needed to guarantee a correct collective behaviour. This shows that such morphisms do not determine interpretations between theories. To support this, the following morphisms are used:

**Definition 3.4.3 (Actor Specification Morphisms)** Given two actor specifications $\Phi_1 = (\Delta_1, \Psi_1)$ and $\Phi_2 = (\Delta_2, \Psi_2)$, a *specification morphism* $\tau : \Phi_1 \to \Phi_2$ is a signature morphism lifted to sentences such that $\vdash_{\Phi_2} \tau(g)$ for every $g \in \Psi_1 \cup Ax_{\Phi_1}$. □

The inclusion of the translated logical axioms $\tau(Ax_{\Phi_1})$ into $\Phi_2$ is necessary as they represent properties which are not always a consequence of $Ax_{\Phi_2}$, since some of these axioms rely on the existence of the original signature symbols only. Once the signature is augmented with new symbols using a morphism, the respective properties may fail to hold. The locality property, for instance, is not preserved by the translation, as shown by Fiadeiro and Maibaum (1992).

Our finite co-completeness result concerning the category of actor signatures easily lifts to categories of extended and parameterised signatures. Much in the same way, it can be transported to a category of actor specifications with the morphisms defined above:

**Proposition 3.4.4 (Category of Actor Specifications)** Actor specifications and morphisms constitute a finitely co-complete category $\mathbf{Spec}^{\mathbf{Act}}$. ■

A comparison between our notion of composability and that of Agha *et al.* (1997) and Talcott (1996b) is in order. As discussed in the previous chapter, composition is realised here by computing co-limits, or pushouts in the particular case of two connected specifications. Given a set of specifications with their pairwise shared sub-components fixed, pushouts of specification morphisms are commutative and have $(\Delta_\perp, \{\,\})$ as their identity. In addition, all their possible compositions in any order are isomorphic among themselves, which yields associativity up to isomorphism. Nevertheless, these are the only similarities with their semantic notion. The composability notion in their work is dynamic and fails to put together components having in common identical names of existing actors. This is syntactically immaterial, though, since there is a canonical way of relating actor syntax and semantics, as hinted by Agha (1986) and followed here, obliging the composed specifications to entail configurations with disjoint sets of existing actor addresses. We treat the dynamic composition of actor components while developing rely-guarantee proofs, as outlined in Section 3.5.

We also need to compare the composition of actor specifications using the morphisms above to the similar usage of categorical notions in Chapter 2. It is particularly important to mention that, because of the implicit parameterisation of actor signatures by a sort of mail addresses and the restricted use of logical action symbols to support interaction, it is not possible to express at the local level any form of extra-logical sharing of signature symbols. This means that at this point interaction is supported logically, always by the synchronised actions introduced in the extension of actor signatures, which may occur simply because the interaction is between actors belonging to the same community or, conversely, because they belong to distinct communities and the designer decided to define morphisms to support their interaction. This is in keeping with the local discipline imposed by the actor model, which precludes any form of interaction other than by object creation and asynchronous message passing.

Using the constructions described above, we can now study communities of heterogeneous actors. A good example is obtained by composing a buffer as described in Section 3.2.1, a processor and a set of terminals to represent a uniprocessor time-sharing architecture. The intended behaviour of the respective component, whose specification shall be called UTSA, is to allow commands

**Actor** TERMINAL

**data types** addr, cmd

**attributes** $bf$ : addr

**actions**

$ter$(addr) : **local + extrn birth**;

$rd$(cmd) : **local computation**;

$tr$(cmd) : **extrn message**

**axioms** $n$ : addr; $v$ : cmd

$ter(n) \to bf = n$ (12.1)

$\mathbf{FG}(\forall v \cdot \neg rd(v))$ (12.2)

$rd(v) \wedge bf = n \to \mathbf{X}(bf = n)$ (12.3)

$rd(v) \wedge bf = n \to \mathbf{X}(\mathbf{send}\ tr, n, v\ ())$ (12.4)

$\mathbf{send}\ tr, n, v\ (\leftarrow) rd(v) \wedge bf = n$ (12.5)

**End**

**Actor** PROCESSOR

**data types** addr, cmd (NEX, BEG : cmd)

**attributes** $in, id$ : addr; $prv$ : cmd

**actions** $pro$(addr, addr) : **local + extrn birth**;

$exc$(int) : **local computation**;

$nop, rec$(int) : **local + extrn message**;

$req$(addr) : **extrn message**

**axioms** $n, p$ : addr; $v$ : cmd

$pro(n,p) \to id = n \wedge in = p \wedge prv = \text{NEX}$ (13.1)

$pro(n,p) \to \mathbf{X}(exc(\text{BEG}) \wedge \mathbf{send}\ nop, n\ ())$ (13.2)

$exc(v) \to \mathbf{X}(prv = v)$ (13.3)

$exc(v) \wedge id = n \wedge in = p \to \mathbf{X}(id = n \wedge in = p)$ (13.4)

$nop \wedge id = n \wedge in = p \to \mathbf{X}(\mathbf{send}\ req, p, n\ ())$ (13.5)

$nop \wedge id = n \to \mathbf{X}(\mathbf{send}\ nop, n\ ())$ (13.6)

$rec(v) \to \mathbf{X}(exc(v))$ (13.7)

$exc(v) \leftarrow rec(v) \vee \exists n, p \cdot pro(n,p) \wedge v = \text{BEG}$ (13.8)

$\mathbf{send}\ nop, n\ (\leftarrow) id = n \wedge \exists p \cdot pro(n,p) \vee nop$ (13.9)

$\mathbf{send}\ req, p, n\ (\leftarrow) nop \wedge id = n \wedge in = p$ (13.10)

$\mathbf{E}(rec(v)) \to v \neq \text{NEX}$ (13.11)

$prv \neq \text{NEX} \to \mathbf{FE}(\mathbf{deliv}\ (nop)) \wedge \mathbf{FE}(nop)$ (13.12)

$prv \neq \text{NEX} \wedge v \neq \text{NEX} \to \mathbf{FE}(\mathbf{deliv}\ (rec(v)))$ (13.13)

$prv \neq \text{NEX} \wedge v \neq \text{NEX} \to \mathbf{FE}(rec(v))$ (13.14)

**End**

Figure 3.2: Simplified specification of terminals and processors.

typed by terminal users to be always processed eventually. The specification of terminal and processor actors for this purpose appear in Figure 3.2.

A terminal becomes aware of the mail address of a cell which will serve as a buffer at creation time (12.1). Afterwards, the terminal always transmits typed commands to this initial buffer cell so that they can wait for processing (12.4). The reading capability of terminals, however, is finite according to (12.2). Processors, in turn, have a more complex behaviour since they have to request commands from the buffer at any possible occasion (13.5). Valid commands may always be eventually delivered to the processor after initialisation (13.13). That is, any command except NEX, which stands for a not executable command, can be delivered to the processor after the first BEG is executed. Once received, any command is subsequently executed (13.7). The computation cycle of the processor alternates among performing no action, in which case time simply passes without witnessing the occurrence of any action, and processing the messages *nop* (13.5, 13.6), *rec* (13.7) or the local computation *exc* (13.3, 13.4). This cycle begins just after the occurrence of the actor birth denoted by the action symbol *pro* (13.2).

Clearly, the actors above cannot work as a single component unless the

(a)

$$\begin{tikzcd}
 & & \mathrm{UTSA} & & \\
 & \textsc{Component1} \arrow[ur, "\kappa_1"] & & \textsc{Component2} \arrow[ul, "\kappa_2"'] & \\
\textsc{Terminal} \arrow[ur, "\tau_{1,1}"] & & \textsc{BufferCell} \arrow[ul, "\tau_{1,2}"'] \arrow[ur, "\tau_{2,1}"] & & \textsc{Processor} \arrow[ul, "\tau_{2,2}"'] \\
 & \textsc{Connector1} \arrow[ul, "\mu_{1,1}"] \arrow[ur, "\mu_{1,2}"'] & & \textsc{Connector2} \arrow[ul, "\mu_{2,1}"] \arrow[ur, "\mu_{2,2}"'] &
\end{tikzcd}$$

(b)

$\textsc{Terminal} \oplus \textsc{BufferCell}$

$$\begin{tikzcd}
tr & x \arrow[l, mapsto, "\mu_{1,1}"'] \arrow[r, mapsto, "\mu_{1,2}"] & put
\end{tikzcd}$$

$\textsc{BufferCell} \oplus \textsc{Processor}$

$$\begin{tikzcd}
get & y \arrow[l, mapsto, "\mu_{2,1}"'] \arrow[r, mapsto, "\mu_{2,2}"] & req \\
reply & z \arrow[l, mapsto, "\mu_{2,1}"'] \arrow[r, mapsto, "\mu_{2,2}"] & rec
\end{tikzcd}$$

(c)

$$\begin{tikzcd}
ter \arrow[r, mapsto, "\kappa_1 \circ \tau_{1,1}"] & terminal \\
pro \arrow[r, mapsto, "\kappa_2 \circ \tau_{2,2}"] & processor \\
nil \arrow[r, mapsto, "\kappa_2 \circ \tau_{2,1}", "\kappa_1 \circ \tau_{1,2}"'] & buffer
\end{tikzcd}$$

Figure 3.3: Static configuration of the multi-tasking system.

proper interconnections between them are provided. Morphisms establish "physical shared channels" to make message passing possible, as defined in Figure 3.3, part (a). COMPONENT1, COMPONENT2 and UTSA, which result from the composition of the three given specifications, are all defined up to isomorphism by the pushout of the given morphisms. This means that any name for each of their symbols suffices as long as the symbols to be shared and only them are equalised. They are defined according to the two connector specifications and the morphisms in Figure 3.3, part (b). The signature of CONNECTOR1 contains one external message symbol only, $x$, which is mapped to the *tr* action of terminals and to the *put* action of buffers. CONNECTOR2 has two such symbols, $y$ and $z$, which are mapped to *get* and *reply* at the buffer side and to *req* and *rec* at the processor side, respectively. The set of axioms in both connector specifications is empty. Assuming that the underlying algebraic morphisms map the mail address sort accordingly and associate integers to commands, the morphisms in the figure clearly satisfy the requirements of Definition 3.4.3.

## 3.5 A Rely-Guarantee Design Discipline

Moving away from the traditional direct approach to specification and verification appears to be inevitable when the features of open reconfigurable systems have to be treated. Isolated specifications establish only the local properties of each specified object. Dynamic (re)configuration and object interaction are two global features which remain completely untreated in this way. The approach for treating such features while preserving the local design discipline is now standard: Chandy and Misra (1981) propose a *rely-guarantee discipline* which allows us to deal with global properties in an organised conditional manner.

Rely-guarantee design is based on the premise that specification and verification take place in a context where the description of component behaviour is relativised to take into account that of the environment, i.e., their limited interaction is described explicitly. Either in specifying or verifying some properties of a component, a rely clause defines a property related to the component which the environment is assumed to satisfy. A guarantee clause is also used to express the properties related to the environment which the component maintains provided that the assumption on the environment holds. The formal semantics of each of these clauses varies according to the adopted design discipline and the mode of interaction assumed in the underlying model. They also provide useful information that may be helpful in some refinement steps (Jones 1983).

Due to our interest in dealing with dynamic configuration and interaction of actor components while preserving the discipline for specification and composition described so far, we choose to impose a rely-guarantee discipline in the verification process only. For a given specification $\Phi$ and finite sets of formulas *init*, *rely*, *pre*, *guar* and *post* based on $\Phi$, we adopt assertions of the form $init : \{pre, rely\}\, \Phi \,\{guar, post\}$ meaning that, given the initial conditions *init*, whenever the pre-conditions *pre* are simultaneously established and the assumptions *rely* are not violated unless the guarantees in *guar* and a post-condition in *post* are obtained, the guarantees are not violated until and necessarily including the moment when the post-condition is obtained, for all the post-conditions in *post*. Putting $J_P \stackrel{\text{def}}{=} \bigwedge\{p|p \in P\}$, such assertions are formalised as follows:

**Definition 3.5.1 (Rely-Guarantee Assertion)** Given a theory presentation $\Phi = (\Delta, \Psi)$ in $\mathsf{obj}$ $\mathbf{Spec}^{\mathbf{Act}}$ and $init \cup rely \cup pre \cup guar \cup post \subseteq \mathcal{G}(\Delta)$ such that each of these sets is finite, *rely-guarantee assertions* are defined below:

**(D16-RG)** $init : \{pre, rely\}\, \Phi \,\{guar, post\} \stackrel{\text{def}}{=}$

$$\Phi \vdash \bigwedge_{p \in post} J_{init} \rightarrow \mathbf{XG}(J_{pre} \wedge (J_{rely})\mathbf{W}(p \wedge J_{guar}) \rightarrow (J_{guar})\mathbf{U}(p \wedge J_{guar})) \quad \Box$$

If compared to other rely-guarantee assertions available in the literature, our definition is rather unusual. The initialisation condition is normally treated elsewhere: Pnueli (1985a) considers that it is provided together with process composition while initialisation is treated as in VDM, separated from the operations, in the extension of this method proposed by Jones (1983). Since we allow any temporal formula in the set of initialisation conditions, they can be used to capture assertion bases, which distinguish communication ports from other variables as proposed by Pandya and Joseph (1991). Rely and pre conditions appear conjoined as an assumption formula in the logical approach advocated by Abadi and Lamport (1995) and by Jonsson and Tsay (1995), much in the way that guarantee and post conditions are conjoined in a commitment formula. The separation adopted here seems to help emphasise the role of each distinct set of properties in the verification process. The distinctions we make are also justified by our desire to use just the rely-guarantee assertions above to deal with dynamic configuration and interaction. A more pragmatic reason justifies the adoption of a set of independently realisable post-conditions. Most authors consider that, if system behaviour or specified operation terminates, this determines a definite state satisfying all the post-conditions (Pandya and Joseph 1991). A *wait* clause is sometimes introduced to express an invariant over the states of non-terminating computations (Cau and Collete 1996). Because we are dealing here with open systems which are never required to terminate but in general eventually validate each of a number of properties, we prefer to adopt post-conditions in the way defined above.

The reader may have correctly observed that with the definition above we have attempted to stay as close as possible to the use of rely-guarantee constructions in model and process based formalisms while taking advantage of the temporal logical features of our own formalism. Let us examine some practical situations that normally arise in designing software systems in this way. First, note that free variables may appear in the formulas of each set of clauses. This is useful, say, in binding actor names to the specifications they comply with. Also note that each set of clauses may be empty. In this case, their respective logical value is equivalent to $\top$, the rely-guarantee assertion is simplified and reasoning becomes easier.

Rely-guarantee constructions are interesting not only due to the additional expressiveness and discipline they introduce into dealing with global phenomena, but also because they may be decomposed and reused. A composition rule is normally proposed to achieve these effects in the verification process. The following inference rule plays this role here:

**Theorem 3.5.2 (Composition Rule)** Given a theory presentation $\Phi = (\Delta, \Psi)$ in obj $\mathbf{Spec^{Act}}$ and $\bigcup\{rely, guar\} \cup \{init_i, pre_i, post_i | 1 \leq i \leq 2\} \subseteq \mathcal{G}(\Delta)$ such that the resulting set is finite, the following inference rule is derivable:

$$
\boxed{
(\mathbf{COMP}) \qquad \frac{
\begin{array}{l}
1.\ \ init_1 : \{pre_1, rely\}\, \Phi\, \{guar, post_1\} \\
2.\ \ init_2 : \{pre_2, guar\}\, \Phi\, \{rely, post_2\}
\end{array}
}{
init_1 \cup init_2 : \{pre_1 \cup pre_2, rely \cup guar\}\, \Phi\, \{rely \cup guar, post_1 \cup post_2\}
}
}
$$

Proof: Assume that $p \in post_1$. We will show that the first premise regarding $p$ can be transformed into a rely-guarantee assertion which complies with the format of the conclusion:

1. $J_{init_1} \rightarrow$ **Ass**, **D16-RG**
   $\mathbf{XG}(J_{pre_1} \wedge (J_{rely})\mathbf{W}(p \wedge J_{guar}) \rightarrow (J_{guar})\mathbf{U}(p \wedge J_{guar}))$
2. $J_{init_1} \wedge J_{init_2} \rightarrow$ **AND-L** 1
   $\mathbf{XG}(J_{pre_1} \wedge (J_{rely})\mathbf{W}(p \wedge J_{guar}) \rightarrow (J_{guar})\mathbf{U}(p \wedge J_{guar}))$
3. $J_{pre_1} \wedge J_{pre_2} \wedge (J_{rely} \wedge J_{guar})\mathbf{W}(p \wedge J_{rely} \wedge J_{guar}) \rightarrow J_{pre_1}$ **REFL**, **AND-L**
4. $(J_{rely} \wedge J_{guar})\mathbf{W}(p \wedge J_{guar}) \rightarrow$ **REFL**, **AND-L**, **R2-G**
   $(J_{rely})\mathbf{W}(p \wedge J_{guar})$ **MON-GW**, **R1-MP** +
5. $(J_{rely} \wedge J_{guar})\mathbf{W}(p \wedge J_{rely} \wedge J_{guar}) \rightarrow$ **REFL**, **AND-L**, **R2-G**
   $(J_{rely} \wedge J_{guar})\mathbf{W}(p \wedge J_{guar})$ **MON-GW**, **R1-MP** +
6. $J_{pre_1} \wedge J_{pre_2} \wedge (J_{rely} \wedge J_{guar})\mathbf{W}(p \wedge J_{rely} \wedge J_{guar}) \rightarrow$ **HS** 5, 4
   $(J_{rely})\mathbf{W}(p \wedge J_{guar})$ **AND-R** +
7. $J_{pre_1} \wedge J_{pre_2} \wedge (J_{rely} \wedge J_{guar})\mathbf{W}(p \wedge J_{rely} \wedge J_{guar}) \rightarrow$ **AND-L** 3, 6
   $J_{pre_1} \wedge (J_{rely})\mathbf{W}(p \wedge J_{guar})$
8. $J_{init_1} \wedge J_{init_2} \rightarrow \mathbf{XG}(J_{pre_1} \wedge J_{pre_2} \wedge$ **RTRAN**, **R1-MP** 7, **R2-G**, **EXP-GX**
   $(J_{rely} \wedge J_{guar})\mathbf{W}(p \wedge J_{rely} \wedge J_{guar}) \rightarrow (J_{guar})\mathbf{U}(p \wedge J_{guar}))$ **R1-MP**, **MON-XG**, **R1-MP**, **HS** 2 +
9. $\mathbf{G}(J_{rely} \wedge J_{guar}) \rightarrow$ **A1-I**, **R2-G**, **MON-G**
   $\mathbf{G}(J_{guar} \rightarrow J_{rely} \wedge J_{guar})$ **R1-MP** +
10. $\mathbf{G}(J_{rely} \wedge J_{guar}) \rightarrow$ **RTRAN**, **R1-MP** 9
    $((J_{guar})\mathbf{U}(p \wedge J_{guar}) \rightarrow (J_{rely} \wedge J_{guar})\mathbf{U}(p \wedge J_{guar}))$ **MON-GU**, **R1-MP** +
11. $\mathbf{G}(p \wedge J_{rely} \wedge J_{guar} \rightarrow p \wedge J_{guar})$ **REFL**, **AND-L**, **R2-G**
12. $(J_{guar})\mathbf{U}(p \wedge J_{rely} \wedge J_{guar}) \rightarrow (J_{guar})\mathbf{U}(p \wedge J_{guar})$ **MON-GU**, **R1-MP** 11
13. $\mathbf{G}(J_{rely} \wedge J_{guar}) \rightarrow$ **PERM**, **R1-MP** 10, **HS** 12
    $((J_{guar})\mathbf{U}(p \wedge J_{rely} \wedge J_{guar}) \rightarrow (J_{rely} \wedge J_{guar})\mathbf{U}(p \wedge J_{guar}))$ **PERM**, **R1-MP** +
14. $J_{rely} \wedge J_{guar} \rightarrow$ **A1-I**, **RTRAN**, **INVE**,
    $(p \wedge J_{guar} \rightarrow p \wedge J_{rely} \wedge J_{guar})$ **HS**, **D4-AND**, **HS** +
15. $\mathbf{G}(J_{rely} \wedge J_{guar}) \rightarrow$ **R2-G** 14, **MON-G**
    $\mathbf{G}(p \wedge J_{guar} \rightarrow p \wedge J_{rely} \wedge J_{guar})$ **R1-MP** +
16. $\mathbf{G}(J_{rely} \wedge J_{guar}) \rightarrow$ **MON-GU**, **HS** 15, **LTRAN**
    $((J_{guar})\mathbf{U}(p \wedge J_{guar}) \rightarrow (J_{rely} \wedge J_{guar})\mathbf{U}(p \wedge J_{rely} \wedge J_{guar}))$ **R1-MP**, **HS** 15 +
17. $(J_{rely} \wedge J_{guar})\mathbf{U}(p \wedge J_{rely} \wedge J_{guar}) \rightarrow$ **A1-I**
    $((J_{guar})\mathbf{U}(p \wedge J_{guar}) \rightarrow (J_{rely} \wedge J_{guar})\mathbf{U}(p \wedge J_{rely} \wedge J_{guar}))$
18. $J_{pre_1} \wedge J_{pre_2} \wedge (J_{rely} \wedge J_{guar})\mathbf{W}(p \wedge J_{rely} \wedge J_{guar}) \rightarrow$ **OR-L** 16, 17
    $((J_{guar})\mathbf{U}(p \wedge J_{guar}) \rightarrow (J_{rely} \wedge J_{guar})\mathbf{U}(p \wedge J_{rely} \wedge J_{guar}))$ **D10-W**, **AND-L** +
19. $J_{init_1} \wedge J_{init_2} \rightarrow \mathbf{XG}(J_{pre_1} \wedge J_{pre_2} \wedge$
    $(J_{rely} \wedge J_{guar})\mathbf{W}(p \wedge J_{rely} \wedge J_{guar}) \rightarrow (J_{rely} \wedge J_{guar})\mathbf{U}(p \wedge J_{rely} \wedge J_{guar}))$

The last step in the derivation above is justified by the following sequence of labels: **A2-I**, **R1-MP** 19, **R2-G**, **EXP-GX**, **R1-MP**, **MON-XG**, **R1-MP**, **HS** 8. Repeating the same process for all the elements of $post_1$ and developing an analogous argument regarding $post_2$, we conclude that the last sentence above is derivable for all the elements of $post_1 \cup post_2$. Note that this process terminates because the involved sets are assumed to be finite. Considering that $J_{P_1} \wedge J_{P_2} \leftrightarrow J_{P_1 \cup P_2}$ for any $P_i \in \{ini_i, pre_i, rely_i, guar_i, post_i\}$, $1 \leq i \leq 2$, an application of **D16** completes the derivation of the conclusion of **COMP**. ■ **(COMP)**

Jonsson and Tsay (1995) have remarked that composition rules such as the above are a simple consequence of the standard meaning of rely-guarantee assertions, which requires the guarantees to be valid when each post-condition is obtained regardless of the validity of the assumptions then. Our Definition 3.5.1 captures this meaning including $J_{guar}$ in the second argument of both **W** and **U**. Note that the first connective is used in the antecedent formula of the defined implication because therein the guarantees and the post-condition cannot be assumed to occur and the definition of unless, **D10-W**, ensures that this is the case. The until connective adopted in the consequent of the same implication says that these formulas eventually obtain, according to **RPL-UF**. The theorem above allows us to infer a more widely applicable composition rule as a corollary:

**Corolary 3.5.3 (General Composition Rule)** Given a theory presentation $\Phi = (\Delta,\ \Psi)$ in obj $\mathbf{Spec^{Act}}$ and the finite set $\bigcup\{init, pre, rely, guar, post\} \cup \{init_i, pre_i, rely_i, guar_i, post_i | 1 \leq i \leq 2\} \subseteq \mathcal{G}(\Delta)$, provided that the following side-conditions are met, the subsequent inference rule is derivable:

$$\begin{array}{ccc} init_1 \cup init_2 \subseteq init & pre_1 \cup pre_2 \subseteq pre & rely_2 \subseteq rely \cup guar_1 \\ guar \subseteq guar_1 \cup guar_2 & post_1 \cup post_2 \subseteq post & rely_1 \subseteq rely \cup guar_2, \end{array}$$

$$\boxed{\begin{array}{ll} \textbf{(GCOMP)} & \begin{array}{ll} 1. & init_1 : \{pre_1, rely_1\}\, \Phi\, \{guar_1, post_1\} \\ 2. & init_2 : \{pre_2, rely_2\}\, \Phi\, \{guar_2, post_2\} \\ \hline & init : \{pre, rely\}\, \Phi\, \{guar, post\} \end{array} \end{array}}$$

Proof: The proof is developed based on the application of **COMP** using the side conditions enumerated above and on the refinement of the given premises using the monotonicity of some temporal connectives. ■ **(GCOMP)**

This rule is more general than that proposed by Cau and Collete (1996) for composing rely-guarantee assertions about synchronous message passing processes because we do not require that each premise refers to a single object only. An analogue to their rule is obtained by providing two connected specifications $\Phi_i$, $1 \leq i \leq 2$, plus the respective morphisms in a way that their pushout determines $\Phi$; $init_i$, $pre_i$, $rely_i$, $guar_i$ and $post_i$ contain only local formulas parameterised by $n_i^1 \in \mathcal{V}_{\mathsf{addr}}$ and $n_i^2$ appears free in $\Phi_i$. Mapping assertions about each specification into $\Phi$ using the given morphisms, **GCOMP** can be applied. Typically, *init* or *pre* would contain a formula like $n^1_i = n^2_{3-i}$, $1 \leq i \leq 2$, to realise the parallel composition of the specified objects. More elaborated rules may deal with hidden symbols which we do not feel necessary here in view of the possibility of organising actor specifications in *design structures* supporting hidden features, in the way originally suggested by Fiadeiro and Maibaum (1994). We will have an opportunity to illustrate the use of these rules and disciplines in Chapter 5.

Now we may conclude the comparison of our constructions and the dynamic algebraic operations for manipulating actor components defined by Talcott (1996b). The simplest such an operation is hiding, which is approximated here by postulating an initial actor configuration using *init* and preventing some objects from receiving messages from the outside environment, using *init* or *rely* depending on whether this constraint is to be static or dynamic. Renaming is another algebraic operation. It does not have a syntactic counterpart but is ensured whenever we refrain from using constants of type addr. Finally, parallel composition is obtained as outlined above. That some components are not composable is reflected here by the impossibility of finding non-empty sets of equalising assumptions of the kind described above while preserving the truth of the antecedent of the resulting assertion. We devote the following section to an example clarifying the verification of rely-guarantee assertions.

## 3.6 Verification of Global Properties

If the uniprocessor time-sharing architecture described in Section 3.4 is to present the behaviour outlined therein, that user commands are always processed eventually, we must stipulate under which circumstances this property is expected. Clearly, there are situations in which this is not established. Assume that a finite number of terminals is connected to a single processor via a buffer. This is the minimal condition we require to ensure that the property above makes sense. Without loss of generality, we postulate that there are exactly two terminals in this configuration. If other arbitrary objects apart from the processor could remove commands from the buffer, if this last component could ignore commands from a specific terminal indefinitely, the characteristic property above would not be established. Considering such properties as part of a rely-guarantee assertion, we can prove that the characteristic property is indeed obtained. Adopting the translations of birth action symbols in Figure 3.3, part (c), the definition $\forall x : y \cdot p[x] \overset{\text{def}}{=} \forall x \cdot Reach(y, x) \rightarrow p[x]$ and a similar one for $\exists$, both based on an auxiliary action symbol *Reach*, we state this assertion as follows:

**Assertion** $C$

| | |
|---|---|
| init | $k.\mathbf{new}(\mathit{buffer}, n), k.\mathbf{new}(\mathit{processor}, m, m, n), k.\mathbf{new}(\mathit{terminal}, t_i, n)\ (1 \leq i \leq 2)$, |
| | $\mathbf{G}(\forall v \cdot \forall y : n \cdot y.put(v) \rightarrow \mathbf{XG}(\neg y.put(v)))$, |
| | $\mathbf{G}(\forall y \cdot (\exists v \cdot y.\mathbf{send}\ put, n, v\ ()) \rightarrow y = t_1 \vee y = t_2)$, |
| | $\mathbf{G}(\forall v \cdot \forall x, y : n \cdot \exists z : x \cdot z = y \vee (\neg x.\mathbf{send}\ put, x.nxt, v\ ())\mathbf{W}(y.\mathbf{send}\ put, y.nxt, v\ ()))$ |
| rely | $\forall y \cdot (\exists x : n \cdot x.get(y)) \rightarrow y = m$ |
| pre | $\exists y \cdot y.rd(v) \wedge (y = t_1 \vee y = t_2), v \neq \mathrm{NEX}$ |
| post | $m.exc(v)$ |

Assertions such as $C$ have the meaning described in the previous section. The first three formulas under init say that the buffer, processor and terminals are considered to be initially created and linked. This illustrates that the designer, in order to be able to verify any global property, is required not only to provide morphisms, allowing actors in different communities to share part of the same language, but also to ensure the existence of some "logical" channels, names which bind actors to each other and enable message passing. Init says in addition that cells of the buffer can only consume each distinct command once, a simplifying assumption, that *put* messages are dispatched to the initial buffer cell $n$ solely by one of the two terminals, and that each cell dispatches a command to the subsequent buffer element only if all the previous cells of the buffer dispatched the same command in the past. The last two properties are static configuration constraints. The dynamic assumption rely on the environment is that the buffer is only requested to send commands to the processor $m$. The formulas under pre and post say that, providing the reading of an executable user command from some terminal, the command is eventually processed. Note that no guarantees are asserted (thus the respective set of formulas is empty) because we are only interested in verifying the post-condition. One guarantee offered by the specified component which we could verify is that all processor requests are addressed to the buffer, a direct consequence of (13.13).

It is important to clarify that the assertion above is expected to be derivable in an extension of UTSA (named $\mathrm{UTSA}_1$) in which the meaning of *Reach* is specified. We adopt the axioms in Figure 3.4. Such auxiliary definitions are conventional in formal methods, especially in model-based formalisms (Jones 1990). Here we have to be careful in using such constructions since our sentences are required to belong to the language of some theory presentation. In turn, the verification of actor component properties often calls for global definitions such as that of *Reach*, which are not allowed by parameterised actor specifications. Moreover, any such auxiliary symbol would possess the additional properties entailed by our actor model axiomatisation. This is not desirable in practice. To overcome this problem, we rely on a functor mapping presentations of our $MSBTL$ extension into this underlying temporal logic. The functor maps

$$\mathbf{beg} \rightarrow \neg \mathit{Reach}(x,y) \quad (3.6.1)$$
$$k.\mathbf{new}(\mathit{nil},x) \vee \exists v \cdot k.\mathbf{new}(\mathit{item},x,v) \rightarrow \mathbf{X}(\mathit{Reach}(x,x)) \quad (3.6.2)$$
$$(k.\mathbf{new}(\mathit{nil},x) \vee \exists v \cdot k.\mathbf{new}(\mathit{item},x,v)) \wedge \mathit{Reach}(y,k) \rightarrow \mathbf{X}(\mathit{Reach}(y,x)) \quad (3.6.3)$$
$$k.\mathbf{new}(\mathit{nil},x) \vee \exists v \cdot k.\mathbf{new}(\mathit{item},x,v) \vee \mathit{Inv}(\mathit{Reach}(x,x)) \quad (3.6.4)$$
$$(k.\mathbf{new}(\mathit{nil},x) \vee \exists v \cdot k.\mathbf{new}(\mathit{item},x,v)) \wedge \mathit{Reach}(y,k) \vee x \neq y \wedge \mathit{Inv}(\mathit{Reach}(y,x)) \quad (3.6.5)$$

Figure 3.4: Definition of *Reach*.

extra-logical axioms via the identity and transforms the logical axioms into sentences of the target theory presentation. Specification morphisms are mapped accordingly. In this way, we can still use our derived inference rules as a valid reasoning technique. Theory presentations allowing definitions as in Figure 3.4 and assertions like $C$ are considered to be in an extension of the functor image. Hereafter, we ignore such technicalities for the sake of simplicity.

What is asserted by $C$ is an instance of the so-called Fair Merge Problem. That is, the processing of sequences of commands from each user must be fair; in other words, that each of them must not have the completion of its execution indefinitely delayed. To understand the validity of this assertion, first note that the respectively linked buffer cells are organised as a reversed queue. Each cell either processes incoming messages or these are forwarded to the remainder of the buffer, because the cell has already been consumed or is not the last element of the queue, or else each message is ignored, because the entire buffer is empty. Now, because the buffer is required by init to receive messages from the two terminals only and these actors eventually stop producing commands according to (12.2), the buffer itself will always be finite in any behaviour, meaning that commands will be fairly stored in and retrieved from this component. Furthermore, since our assumption rely is that the buffer is hidden from the environment with respect to receiving *get* messages, only the processor will recurrently request commands and possibly receive a reply from the buffer. Each command dispatched by a terminal will be eventually processed in this way.

The explanation above does not clarify how the formulas in our assumption were chosen nor the criteria for their placement in one clause or another. The init clause should only contain formulas describing the initial state and the static configuration constraints that always hold about the component. The pre-conditions should just trigger the eventual occurrence of each post-condition. Rely formulas are expected to be true until but not necessarily including each of these post-condition occurrences. Note that we could have written our penultimate initialisation condition as part of our rely assertion. However, such an assertion would be too weak for our purposes: those messages dispatched before the occurrence of the pre-condition could produce undesirable interference in the behaviour of the architecture. On the other hand, reducing assumptions to the local delivery of messages, as we did in proposing the rely condition, corresponds to ground our analyses on the "local time" of each component (Clinger 1981).

The rigorous verification of the assertion above is based on the definition of a (relative) well-founded relation as explained in Section 2.7. We propose an extension of $\mathrm{UTSA}_1$ (named $\mathrm{UTSA}_2$) and choose to include in this extension the following action symbol definition concerning buffer cells:

$$R(n,x,y) \leftrightarrow \mathit{Reach}(n,x) \wedge \mathit{Reach}(n,y) \wedge y.\mathit{nxt} = x \wedge y.\mathit{lst} = \mathrm{F} \tag{3.6.6}$$

In general, $R$ does not define a well-founded relation. If we take into account just those cells reachable from the assumed initial buffer element $n$, a well-founded relation is indeed defined. To verify this, we consider in the sequel that $\mathrm{UTSA}_2$ is also endowed with an unconstrained flexible symbol $t$ of sort addr and that formulas in each of the clauses of our assertion are linearly ordered according to their position in $C$ to facilitate references. We omit almost all the details which are not strictly necessary for comprehension and perform the verification in a relativised context, considering that all the derived sentences are always valid strictly after the occurrence of the initialisation condition. We are allowed to reason in this way provided that we refrain from using temporal logical inference rules. Note that most of our derived inference rules are relative to the occurrence of the initialisation condition. Relativised rules similar to those for introducing unconstrained flexible symbols and well-founded induction can also be shown admissible.

The two static characteristic properties of flexible well-founded relations, irreflexivity and stability, are verified as follows:

**[IRR]** $\forall x \cdot \neg R(n,x,x)$

Considering that $\neg R(n,x,x) \leftrightarrow \neg \mathit{Reach}(n,x) \vee x.\mathit{nxt} \neq x \vee x.\mathit{lst} = \mathrm{T}$, we sketch this proof as follows, abbreviating as $\mathit{CREATE}(x,y)$ the formula $\exists v \cdot x.\mathbf{new}(\mathit{item},y,v) \vee x.\mathbf{new}(\mathit{nil},y)$:

1. $x.\mathit{nil} \rightarrow x.\mathit{nxt} \neq x \vee x.\mathit{lst} = \mathrm{T}$ (11.1)
2. $x.\mathit{item}(v) \rightarrow x.\mathit{nxt} \neq x \vee x.\mathit{lst} = \mathrm{T}$ (11.2)
3. $x.\mathit{go} \wedge (x.\mathit{nxt} \neq x \vee x.\mathit{lst} = \mathrm{T}) \rightarrow \mathbf{X}(x.\mathit{nxt} \neq x \vee x.\mathit{lst} = \mathrm{T})$ (11.5)
4. $x.\mathit{cons} \wedge (x.\mathit{nxt} \neq x \vee x.\mathit{lst} = \mathrm{T}) \rightarrow \mathbf{X}(x.\mathit{nxt} \neq x \vee x.\mathit{lst} = \mathrm{T})$ (11.6)
5. $x.\mathit{link}(y) \leftrightarrow \exists v \cdot x.\mathbf{new}(\mathit{item},y,v)$ (11.13), (11.9)

| | | |
|---|---|---|
| 6. | $\exists v \cdot x.\mathbf{new}(item, x, v) \rightarrow \exists v \cdot x.\mathbf{new}(item, x, v)$ | **REFL** |
| 7. | $\exists v \cdot x.\mathbf{new}(item, x, v) \rightarrow \nexists v \cdot x.\mathbf{new}(item, x, v) \vee CREATE(x, x)$ | **OR-R** 6 |
| 8. | $\exists v \cdot x.\mathbf{new}(item, x, v) \rightarrow \mathbf{XG}\bot$ | **EXIST** 6, 7 |
| 9. | $\exists v \cdot x.\mathbf{new}(item, x, v) \rightarrow \bot$ | 8, **REFL-G**, **A11-X** |
| 10. | $\exists v \cdot x.\mathbf{new}(item, y, x) \rightarrow x \neq y$ | 9, **A22-EQ** |
| 11. | $\exists v \cdot x.\mathbf{new}(item, y, x) \rightarrow \mathbf{X}(x \neq y)$ | 10, **RIGID** |
| 12. | $x.link(y) \rightarrow \mathbf{X}(x \neq y)$ | **IFF-E** 5, **HS** 11 |
| 13. | $x.link(y) \wedge (x.nxt \neq x \vee x.lst = \mathrm{T}) \rightarrow \mathbf{X}(x.nxt \neq x \vee x.lst = \mathrm{T})$ | 12, (11.7) |
| 14. | $x.nxt \neq x \vee x.lst = \mathrm{T}$ | **SAFE** 1-4, 13 |
| 15. | $\neg Reach(n, x) \vee x.nxt \neq x \vee x.lst = \mathrm{T}$ | **A1-I**, 14, **D3-OR** |

**[STAB]** $\forall x, y \cdot R(n, x, y) \rightarrow \mathbf{X}(R(n, x, y))$

The reader is asked to work out the full proof. Here we just outline the most important proof steps:

$$\begin{aligned}
R(n,x,y) \quad &\overset{(3.6.6)}{\rightarrow} \quad Reach(n,x) \wedge Reach(n,y) \wedge y.nxt = x \wedge y.lst = \mathrm{F} \\
&\overset{(3.3.1)}{\rightarrow} \quad Reach(n,x) \wedge Reach(n,y) \wedge y.nxt = x \wedge y.lst = \mathrm{F} \wedge \mathbf{G}(\neg link(k)) \\
&\overset{\substack{\mathbf{INV} \\ (3.6.1-5)}}{\rightarrow} \quad \mathbf{G}(Reach(n,x) \wedge Reach(n,y) \wedge y.nxt = x \wedge y.lst = \mathrm{F}) \\
&\overset{(3.6.6)}{\rightarrow} \quad \mathbf{X}(R(n,x,y))
\end{aligned}$$

To verify the two dynamic properties of well-founded relations, change termination and anti-progressiveness, we have to develop a number of auxiliary results first. We begin by showing that buffer cells reachable from $n$ obey a causal law which prevents them from being at the same time reachable from and related by $R$ to any other fixed cell (**CAUSAL**). Next we show that *Reach* as defined in Figure 3.4 determines a transitive relation (**T-***Reach*). We also prove that the directed binary relation determined by $R$ with first argument fixed on $n$ is acyclic (**ACYCLIC**). Then we proceed with the verification of **TERM** and **APROG**.

**(CAUSAL)** $\forall z : n \cdot \forall y : z \cdot \neg R(n, z, y)$

First note that the following two sentences are true as a consequence of the axioms in Figure 3.1 and 3.4:

$$\mathbf{beg} \rightarrow ((\neg Reach(z,z))\mathbf{W}(\neg Reach(z,y)))\mathbf{W}(\exists k : n \cdot CREATE(k,y))$$
$$\mathbf{beg} \rightarrow (\neg R(n,z,y))\mathbf{W}(y.\mathbf{new}(item,z) \wedge Reach(n,y))$$

Assume that $\mathit{Reach}(z, y) \wedge R(n, z, y)$ is the case. Hence, $k$ above has to be equal to $y$, but in this case there would exist two actors $z$ and $y$ which create each other and the resulting temporal paradox contradicts a consequence of **O7a** and **EXIST**. Generalising the negation of our assumption to all the other cells reachable from $n$ clearly implies **CAUSAL**.

**(T-*Reach*)** $\forall x, y, z \cdot \mathit{Reach}(x, y) \rightarrow (\mathit{Reach}(y, z) \rightarrow \mathit{Reach}(x, z))$

We write the body of the quantified formula above as $\mathit{TRANS}(x, y, z)$. This sentence is verified using **IND-begG**:

| | | |
|---|---|---|
| 1. | $\mathbf{beg} \rightarrow (\mathit{Reach}(y, z) \rightarrow \mathit{Reach}(x, z))$ | (3.6.1), **NEG-L**, **PERM** |
| 2. | $\mathbf{beg} \rightarrow \forall x, y, z \cdot \mathit{TRANS}(x, y, z)$ | 1, **A1-I**, **HS**, **R5-∀** |
| 3. | $\exists k \cdot \mathit{CREATE}(k, x) \rightarrow (\mathit{Reach}(x, x) \rightarrow \mathbf{X}(\mathit{Reach}(x, x)))$ | (3.6.2) |
| 4. | $\nexists k \cdot \mathit{CREATE}(k, x) \rightarrow (\mathit{Reach}(x, x) \rightarrow \mathbf{X}(\mathit{Reach}(x, x)))$ | (3.6.3) |
| 5. | $\mathit{Reach}(x, x) \rightarrow \mathbf{X}(\mathit{Reach}(x, x))$ | 3, 4 |
| 6. | $\mathit{CREATE}(x, y) \wedge \mathit{Reach}(z, x) \rightarrow$ $(z \neq x \wedge \mathit{Reach}(z, y) \rightarrow \mathbf{X}(\mathit{Reach}(z, y)))$ | (3.6.4) |
| 7. | $\neg(\mathit{CREATE}(x, y) \wedge \mathit{Reach}(z, x)) \rightarrow$ $(z \neq x \wedge \mathit{Reach}(z, y) \rightarrow \mathbf{X}(\mathit{Reach}(z, y)))$ | (3.6.5) |
| 8. | $z \neq x \wedge \mathit{Reach}(z, x) \rightarrow \mathbf{X}(\mathit{Reach}(z, x))$ | 6, 7 |
| 9. | $\forall x, y, z \cdot \mathit{TRANS}(x, y, z) \rightarrow \mathbf{X}(\forall x, y, z \cdot \mathit{TRANS}(x, y, z))$ | 5, 8 |
| 10. | $\mathbf{G}(\forall x, y, z \cdot \mathit{TRANS}(x, y, z) \rightarrow \mathbf{X}(\forall x, y, z \cdot \mathit{TRANS}(x, y, z)))$ | **R2-G** 9 |
| 11. | $\forall x, y, z \cdot \mathit{TRANS}(x, y, z)$ | **IND-begG** 2, 10 |

**(ACYCLIC)** $\forall x, z : n \cdot \forall y : x, w : z \cdot R(n, z, y) \rightarrow w \neq x$

| | | |
|---|---|---|
| 1. | $\forall z : n \cdot \forall y : z \cdot \neg R(n, z, y)$ | **CAUSAL** |
| 2. | $\forall z : n \cdot \forall y \cdot \mathit{Reach}(z, y) \rightarrow \neg R(n, z, y)$ | 1 |
| 3. | $\forall z : n \cdot \forall x, y \cdot \mathit{Reach}(x, y) \rightarrow (\mathit{Reach}(z, x) \rightarrow \neg R(n, z, y))$ | 2, **T-*Reach*** |
| 4. | $\forall z : n \cdot \forall x, y \cdot \forall w \cdot w = x \rightarrow (\mathit{Reach}(x, y) \rightarrow (\mathit{Reach}(z, w) \rightarrow \neg R(n, z, y)))$ | 3, **A22-EQ** |
| 5. | $\forall z : n \cdot \forall x, y \cdot \forall w \cdot \mathit{Reach}(x, y) \rightarrow (\mathit{Reach}(z, w) \rightarrow (w = x \rightarrow \neg R(n, z, y)))$ | 4, **PERM** |
| 6. | $\forall z : n \cdot \forall x, y \cdot \forall w \cdot \mathit{Reach}(x, y) \rightarrow (\mathit{Reach}(z, w) \rightarrow (R(n, z, y) \rightarrow w \neq x))$ | 5, **INVE** |
| 7. | $\forall z : n \cdot \forall x, y \cdot \mathit{Reach}(x, y) \rightarrow \forall w : z \cdot R(n, z, y) \rightarrow w \neq x$ | 6, **MOV-IF∀** |
| 8. | $\forall z : n \cdot \forall x \cdot \forall y : x \cdot \forall w : z \cdot R(n, z, y) \rightarrow w \neq x$ | 7 |
| 9. | $\forall x, z : n \cdot \forall y : x, w : z \cdot R(n, z, y) \rightarrow w \neq x$ | 8, **A1-I** |

**[TERM]** $\mathbf{FG}(\forall x, y \cdot \neg R(n, x, y) \rightarrow \mathbf{X}(\neg R(n, x, y)))$

First note that substituting $p_1$ and $q$ by $\top$ and $p_2$ by $\mathsf{init}_{123}\text{-}C$ in both **NRESP** and **NCOM**, we obtain as a consequence the following sentences:

$$\mathsf{init}_{123}\text{-}C \rightarrow \mathbf{XG}(\mathbf{FG}(\nexists x \cdot x.\mathbf{send}\ \mathit{put}, n, v\ ()) \rightarrow \mathbf{FG}(\neg x.\mathbf{deliv}\ (\mathit{put}, v))) \quad (3.6.7)$$

$$\mathsf{init}_{123}\text{-}C \rightarrow \mathbf{XG}(\mathbf{FG}(\neg x.\mathbf{deliv}\ (\mathit{put}, v)) \rightarrow \mathbf{FG}(\neg x.\mathit{put}(v))) \quad (3.6.8)$$

Linking these two sentences and relying on $\mathsf{init}_4$-$C$ and $\mathsf{init}_6$-$C$, we obtain after $\mathsf{init}_{123}$-$C$:

$$\mathbf{FG}(\nexists x \cdot x.\mathbf{send}\ put, n, v\ ()) \rightarrow \forall y : n \cdot \mathbf{FG}(\nexists x \cdot x.\mathbf{send}\ put, y, v\ ()) \qquad (3.6.9)$$

We proceed with the following derivation:

| | | |
|---|---|---|
| 1. | $\mathbf{FG}(\forall v \cdot \neg t_1.rd(v))$ | (12.2), $\mathsf{init}_3$-$C$ |
| 2. | $\mathbf{FG}(\forall v \cdot \neg t_1.\mathbf{send}\ tr, n, v\ ())$ | 1, (12.5) |
| 3. | $\mathbf{FG}(\forall v \cdot \neg t_2.rd(v))$ | (12.2), $\mathsf{init}_3$-$C$ |
| 4. | $\mathbf{FG}(\forall v \cdot \neg t_2.\mathbf{send}\ tr, n, v\ ())$ | 3, (12.5) |
| 5. | $\mathbf{FG}(\forall v \cdot \neg t_1.\mathbf{send}\ tr, n, v\ (\wedge) \neg t_2.\mathbf{send}\ tr, n, v\ ())$ | 2, 4, **DIST-ANDFG** |
| 6. | $\mathbf{FG}(\forall x, v \cdot \neg x.\mathbf{send}\ put, n, v\ ())$ | 5, $\mathsf{init}_5$-$C$ |
| 7. | $\forall y : n \cdot \mathbf{FG}(\forall x, v \cdot \neg x.\mathbf{send}\ put, y, v\ ())$ | 6, (3.6.9) |
| 8. | $\forall y : n \cdot \mathbf{FG}(\forall v \cdot \neg y.\mathbf{deliv}\ (put, v))$ | 7, **O9** |
| 9. | $\forall y : n \cdot \mathbf{FG}(\forall v \cdot \neg y.put(v))$ | 8, **O11** |
| 10. | $\forall y : n \cdot \mathbf{FG}(\forall v \cdot \neg y.link(v))$ | 9, (11.13) |
| 11. | $\forall y : n \cdot \mathbf{FG}(\forall x \cdot y.nxt \neq x \vee y.lst = \mathrm{T} \rightarrow \mathbf{X}(y.nxt \neq x \vee y.lst = \mathrm{T}))$ | (11.5), (11.6) |
| | | 10, **INV** + |
| 12. | $\forall y : n \cdot \mathbf{FG}(\forall x \cdot \neg Reach(y, x) \rightarrow \mathbf{X}(\neg Reach(y, x)))$ | (11.9), (11.13) |
| | | 10, (3.6.5) + |
| 13. | $\neg R(n, x, y) \leftrightarrow \neg Reach(n, x) \vee \neg Reach(n, y) \vee y.nxt \neq x \vee y.lst = \mathrm{T}$ | (3.6.6) |
| 14. | $\mathbf{FG}(\forall x, y \cdot \neg R(n, x, y) \rightarrow \mathbf{X}(\neg R(n, x, y)))$ | 11, 12, 13, **T-***Reach* |
| | | **BARC-FG** |

**[APROG]** $\mathbf{G}(\forall x \cdot t = x \rightarrow \mathbf{X}(t = x \vee R(n, t, x))) \rightarrow \mathbf{FG}(\forall x \cdot t = x \rightarrow \mathbf{X}(t = x))$

Suppose that the antecedent of the implication above is the case but the consequent of the same formula is not. This assumption implies

$$\mathbf{GF}(\exists x \cdot t = x \wedge \mathbf{X}(R(n, t, x)))$$

As a result, since $R$ with first argument fixed on $n$ is acyclic, the value of $t$ can only forever eventually decrease. This generates a contradiction with **STAB** and **TERM**, which say that the same relation eventually stops changing and then relates just a finite number of mail addresses reachable from $n$. ■

Now that we know $R$ defines a well-founded relation relative to the initial buffer cell $n$, we can continue the verification of our rely-guarantee assertion applying our relativised version of the inference rule **WELL**. As it turns out, however, we are obliged to develop a number of auxiliary results to support an application of such a rule. Since the processor is the most "active" component of the specified architecture, we examine the properties of this actor first. We show below that each previously processed command can never be the non-executable one (NEX) after the processor becomes live:

| | | |
|---|---|---|
| 1. | $v = \mathrm{NEX} \rightarrow \mathbf{A}(\neg m.rec(v))$ | (13.11), **INVE**, **DUAL-AE** |
| 2. | $\neg m.rec(\mathrm{NEX})$ | 1, **REFL-A**, **A22-EQ** |
| 3. | $\neg m.exc(\mathrm{NEX})$ | 2, (13.8), cmd $\boldsymbol{Ax}$ |
| 4. | $m.exc(v) \wedge m.prv \neq \mathrm{NEX} \rightarrow \mathbf{X}(m.prv \neq \mathrm{NEX})$ | 3, (13.3) |
| 5. | $m.prv \neq \mathrm{NEX} \rightarrow \mathbf{G}(m.prv \neq \mathrm{NEX})$ | 4, **INV** |
| 6. | $m.pro(n, p) \rightarrow \mathbf{F}(m.prv \neq \mathrm{NEX})$ | (13.2), (13.3) |
| 7. | $\mathbf{FG}(m.prv \neq \mathrm{NEX})$ | 6, 5, init$_2$-$C$, **O7a** |

The fact that invalid commands are (eventually) never executed is important because this is an invariant property of the processor which enables the delivery and consumption of messages. This property allows us to show that any executable command dispatched to the processor is eventually executed:

| | | |
|---|---|---|
| 8. | $m.rec(v) \rightarrow \mathbf{F}(m.exc(v))$ | (13.7) |
| 9. | $m.prv \neq \mathrm{NEX} \wedge v \neq \mathrm{NEX} \rightarrow \mathbf{FE}(m.rec(v))$ | (13.14) |
| 10. | $m.\mathbf{deliv}\ (rec, v) \rightarrow$ $\mathbf{X}(\mathbf{F}(m.prv \neq \mathrm{NEX} \wedge v \neq \mathrm{NEX}) \rightarrow \mathbf{F}(m.exc(v)))$ | **RESP** 4, 8, 9 |
| 11. | $\mathbf{XF}(m.prv \neq \mathrm{NEX})$ | 7, **COM-FG**, **RPL-GX** |
| 12. | $m.\mathbf{deliv}\ (rec, v) \rightarrow \mathbf{F}(v \neq \mathrm{NEX} \rightarrow m.exc(v))$ | 11, 10, **MON-X**, **RIGID** |
| 13. | $m.prv \neq \mathrm{NEX} \wedge v \neq \mathrm{NEX} \rightarrow \mathbf{FE}(m.\mathbf{deliv}\ (rec, v))$ | (13.13) |
| 14. | $\exists k \cdot k.\mathbf{send}\ rec, m, v\ (\rightarrow)$ $\mathbf{X}(\mathbf{F}(m.prv \neq \mathrm{NEX} \wedge v \neq \mathrm{NEX}) \rightarrow \mathbf{F}(v \neq \mathrm{NEX} \rightarrow m.exc(v)))$ | **COM** 4, 12, 13 |
| 15. | $\exists k \cdot k.\mathbf{send}\ reply, m, v\ (\wedge) v \neq \mathrm{NEX} \rightarrow \mathbf{F}(m.exc(v))$ | 11, 14, **MON-X**, **RIGID** |

Another important property exhibited by the processor is that *nop* messages are always eventually self-dispatched and consumed. As a result, the processor keeps requesting commands from the buffer regularly:

| | | |
|---|---|---|
| 16. | $m.exc(v) \wedge m.id = x \wedge m.in = y \rightarrow \mathbf{X}(m.id = x \wedge m.in = y)$ | (13.4) |
| 17. | $m.id = x \wedge m.in = y \rightarrow \mathbf{G}(m.id = x \wedge m.in = y)$ | **INV** 16 |
| 18. | $\mathbf{G}(m.id = m \wedge m.in = n)$ | 17, (13.1), init$_2$-$C$ |
| 19. | $m.nop \rightarrow \mathbf{F}(m.nop)$ | **REFL**, **D8-F** |
| 20. | $m.prv \neq \mathrm{NEX} \rightarrow \mathbf{FE}(m.nop)$ | (13.12) |
| 21. | $m.\mathbf{deliv}\ (nop) \rightarrow \mathbf{X}(\mathbf{F}(m.prv \neq \mathrm{NEX}) \rightarrow \mathbf{F}(m.nop))$ | **RESP** 4, 19, 20 |
| 22. | $m.\mathbf{deliv}\ (nop) \rightarrow \mathbf{F}(m.nop)$ | 11, 21, **MON-X** |
| 23. | $m.prv \neq \mathrm{NEX} \rightarrow \mathbf{FE}(m.\mathbf{deliv}\ (nop))$ | (13.12) |
| 24. | $m.\mathbf{send}\ nop, m\ (\rightarrow)\mathbf{X}(\mathbf{F}(m.prv \neq \mathrm{NEX}) \rightarrow \mathbf{F}(m.nop))$ | **COM** 4, 22, 23 |
| 25. | $m.\mathbf{send}\ nop, m\ (\rightarrow)\mathbf{XF}(m.nop)$ | 11, 24, **MON-X** |
| 26. | $m.nop \rightarrow \mathbf{XF}(m.nop)$ | 18, 25, (13.6) |
| 27. | $\mathbf{F}(m.nop) \rightarrow \mathbf{GF}(m.nop)$ | 26, **IDEM-G**, **IND-G** |
| 28. | $\mathbf{GF}(m.nop)$ | 27, (13.2), init$_2$-$C$ |
| 29. | $\mathbf{GF}(m.\mathbf{send}\ req, n, m\ ())$ | 18, 28, (13.5) |

We wish to apply our well-founded induction rule using the new relation symbol $R$ to prove that valid commands produced in one of the two terminals and dispatched to the buffer are eventually processed. Two other properties are required to ensure that the connections between terminals and buffer and between this last component and the processor present the expected behaviour:

$$m.\mathbf{send}\ \mathit{req}, n, m\ (\rightarrow)\mathbf{F}(n.\mathbf{deliv}\ (\mathit{get}, m)) \tag{3.6.10}$$

$$t_i.\mathbf{send}\ \mathit{tr}, n, v\ (\rightarrow)\mathbf{F}(n.\mathbf{deliv}\ (\mathit{put}, v)) \tag{3.6.11}$$

The omitted verification of these properties is analogous to that of (15).

Because we know that each typed command is eventually delivered to the buffer (3.6.11), the processor is always eventually producing new command requests (29), the requests are eventually delivered to the buffer (3.6.10), and that valid commands dispatched as a result by the buffer are eventually processed (15), we can develop our inductive argument taking only into account the buffer specification. We propose an inductive assertion saying that, whenever a valid command is delivered to a buffer cell $x$ ($d_{ind}[x]$) and the same cell always eventually receives requests from the processor ($r_{ind}[x]$), provided that just the processor is allowed to consume the contents of such buffer cell or its sucessor ($c_{ind}[x]$), in the future either there is a cell in the buffer dispatching the newly deposited command to the processor ($q_{ind}$) or there is another cell $y$ related to $x$ for which the same property obtains ($p_{ind}[y]$). Applying our induction rule, we reach a conclusion which, when connected to the properties mentioned above, corresponds to our rely-guarantee assertion. We use the following abbreviations to write our inductive assertion:

$$\begin{aligned}
d_{ind}[x] &\overset{\text{def}}{=} \mathit{Reach}(n, x) \wedge x.\mathbf{deliv}\ (\mathit{put}, v) \wedge v \neq \text{NEX} \\
r_{ind}[x] &\overset{\text{def}}{=} \mathbf{GF}(x.\mathbf{deliv}\ (\mathit{get}, m)) \\
c_{ind}[x] &\overset{\text{def}}{=} (\forall y \cdot (x.\mathit{get}(y) \vee x.\mathit{nxt}.\mathit{get}(y) \wedge x.\mathit{lst} = \text{F}) \rightarrow y = m)\mathbf{W}(q_{ind}) \\
p_{ind}[x] &\overset{\text{def}}{=} d_{ind}[x] \wedge r_{ind}[x] \wedge c_{ind}[x] \\
q_{ind} &\overset{\text{def}}{=} \exists k : n \cdot k.\mathbf{send}\ \mathit{reply}, m, v\ (\wedge) v \neq \text{NEX}
\end{aligned}$$

In this way, our induction assertion becomes:

$$\forall x \cdot p_{ind}[x] \rightarrow \mathbf{F}(q_{ind} \vee \exists y \cdot R(n, y, x) \wedge p_{ind}[y]) \tag{3.6.12}$$

Let us informally justify the derivability of (3.6.12). First note that, because of (11.9) and (11.10), whenever a *put* message is delivered and subsequently consumed by a buffer cell $x$, either the current cell is the last in the buffer and in this case a new one is created to store the message contents therein or the message is forwarded to the subsequent buffer element. In the former base case, considering that what has just been stored in the buffer is a valid command, because $x$ is assumed to always eventually receive *get* messages from the processor, according to $r_{ind}[x]$, and can only process one such *get* message, by $c_{ind}[x]$ and the last step in the verification of (3.3.1) in Section 3.3, the first of these messages will consume the command $v$ or alternatively the subsequent cell will have the same contents consumed sometime in the future, since the next cell eventually satisfies both requirements. In either case, $q_{ind}$ is obtained. Otherwise, if the cell is not the last in the buffer, (11.10) guarantees that the next existing cell $y$ related to $x$ will eventually satisfy $p_{ind}[y]$. Assuming that $x$ is reachable from $n$, it is not difficult to prove using the axioms in Figure 3.4 that $n$, $x$ and $y$ are related by $R$.

The application of our well-founded relation rule to (3.6.12) yields the following sentence:

$$(\exists x \cdot p_{ind}[x]) \rightarrow \mathbf{F}(q_{ind}) \tag{3.6.13}$$

First of all, note that $p_{ind}[n]$ implies $\exists x \cdot p_{ind}[x]$. For any $v \neq \text{NEX}$, (3.6.11) implies $d_{ind}[n]$. In addition, $\mathsf{rely}_{12}\text{-}C$ leads to $c_{ind}[n]$ and the formula $r_{ind}[n]$ is a consequence of the proof step (29) above when connected to (3.6.10). Finally, $q_{ind}$ implies $\mathsf{post}\text{-}C$ when connected to (11). Putting these considerations together and reintroducing our initialisation condition, we conclude the verification of $C$. ■

As a final observation in this section, it is worthwhile mentioning that the necessity of using temporal sentences to establish in a (pseudo)-finitary manner the well-foundedness of binary relations within first-order temporal logic reinforces the point of view that complex dynamic data types such as lists and queues should be treated as first class objects (Agha 1986, Milner *et al.* 1992). Using our formalism, it would not be possible to perform otherwise any kind of inductive reasoning over their structure in order to verify liveness properties.

## 3.7 A Plethora of Modes of Interaction

Apart from the asynchronous mode of message transmission assumed in the actor model, components of real distributed systems may also interact through point-to-point message passing modes which require more synchrony. Charron-Bost *et al.* (1996) define a hierarchy of increasingly more synchronous interaction modes, where *FIFO communication* is mentioned as an example in which the messages exchanged between each two components must be received in the order

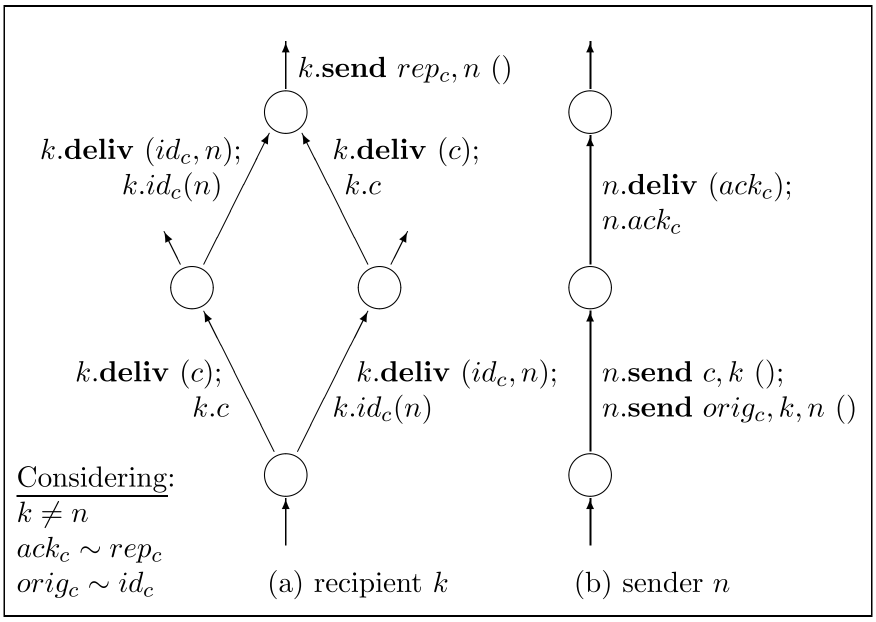

$k.\mathbf{send}\ rep_c, n\ ()$
$k.\mathbf{deliv}\ (id_c, n);$
$k.id_c(n)$
$k.\mathbf{deliv}\ (c);$
$k.c$
$n.\mathbf{deliv}\ (ack_c);$
$n.ack_c$
$k.\mathbf{deliv}\ (c);$
$k.c$
$k.\mathbf{deliv}\ (id_c, n);$
$k.id_c(n)$
$n.\mathbf{send}\ c, k\ ();$
$n.\mathbf{send}\ orig_c, k, n\ ()$
Considering:
$k \neq n$
$ack_c \sim rep_c$
$orig_c \sim id_c$
(a) recipient $k$
(b) sender $n$

Figure 3.5: Protocol for ensuring synchrony of reconfigurable objects.

they are sent. We show in this section that these other modes of interaction can also be captured in terms of actors. In this way, we shall be able to conclude that the asynchrony assumption does not really restrict expressive power in designing open reconfigurable systems.

In order to illustrate in a direct way how to support other modes of interaction in terms of the actor model, we would simply have to provide a set of example specifications describing objects that behave accordingly. Instead, we prefer to adopt a standard generic technique in distributed systems theory defining transformations which, when applied to a complex description like our architecture specification in Section 3.6, guarantee that the resulting description ensures the required property. See Liu and Joseph (1992) for transformations ensuring fault tolerance and Hadzilacos and Toueg (1994) for transformations in the mode of interaction of broadcasting programs. To exemplify this technique, we choose to address here only the full synchrony case, since the definition of actor specification transformations may have to be quite elaborate — in the case of FIFO communication, for example, we could choose to associate messages to sequence numbers. Moreover, because our research is concerned only with software design, we stick to this level of abstraction dealing with specifications as base subject of transformation, in contrast to the programs used in the aforementioned work. We also adopt the same categorical techniques of previous sections.

Objects that interact via asynchronous point-to-point message passing can be transformed into synchronous ones by obliging each dispatched message to be acknowledged and by forcing the originating object to stay in a wait state until such an acknowledgement is received. Then, the normal behaviour can be resumed. However, because we consider reconfigurable systems here, some additional treatment is required to inform the recipient of each message about the originating object mail address, to prevent the sender from automatically deadlocking or dispatching unsolicited responses if self-addressed messages are dispatched. One way of treating the first problem while preserving the original specification is to force the recipient of each message to enter into an auxiliary state which is abandoned only if both message and originator address are received. This behaviour is illustrated by the diagram in Figure 3.5.

We formalise the synchrony transformation in terms of actor specifications and morphisms. This is done by defining an extension of each given signature with the additional symbols in Figure 3.5 and each set of axioms with the synchronisation properties described above. The following definition captures this transformation:

**Definition 3.7.1 (Synchrony transformation)** Given a actor specification $\Phi_1 = (\Delta_1, \Psi_1)$ in obj $\mathbf{Spec}^{\mathbf{Act}}$, a specification morphism $\Phi_1 \xrightarrow{\sigma} \Phi_2$ in morph $\mathbf{Spec}^{\mathbf{Act}}$ obeying the following conditions is said to represent a *synchrony transformation*: The signature $\Delta_2$ is determined by the $\Delta_1$-image under $\sigma$ and the following conditions:

1. $\Sigma_2 = (\sigma(S_1) \cup \{\mathsf{bool}\}, \sigma(\Omega_1) \cup \{\mathrm{T}_{\mathsf{bool}}, \mathrm{F}_{\mathsf{bool}}, \mathrm{NOT}_{\mathsf{bool}\to\mathsf{bool}}\})$;

2. For each symbol $c \in \Gamma_{e_1 - e_{b_1}}$, $\mathit{type}(c) = \langle s_1, \ldots, s_n\rangle$, there are $\mathit{ack}_{\sigma(c)} \in \Gamma_{(l_2 - l_{b_2}) \cap (e_2 - e_{b_2})}$ and $\mathit{switch}_{\sigma(c)} \in \Gamma_{c_2}$ of type $\langle \mathsf{addr}, \sigma(s_1), \ldots, \sigma(s_n)\rangle$; and $\mathit{wait}_{\sigma(c)} \in \mathcal{A}_2$ such that $\mathit{type}(\mathit{wait}_{\sigma(c)}) = \langle \mathsf{addr}, \sigma(s_1), \ldots, \sigma(s_n)\rangle \to \mathsf{bool}$;

3. For each $c \in \Gamma_{l_1 - l_{b_1}}$, $\mathit{type}(c) = \langle s_1, \ldots, s_n\rangle$, there are $\{\mathit{rep}_{\sigma(c)}, \mathit{id}_{\sigma(c)}\} \in \Gamma_{(l_2 - l_{b_2}) \cap (e_2 - e_{b_2})}$ and $\mathit{rid}_{\sigma(c)} \in \Gamma_{c_2}$, all of these of type $\langle \mathsf{addr}, \sigma(s_1), \ldots, \sigma(s_n)\rangle$; $\mathit{rmsg}_{\sigma(c)} \in \Gamma_{c_2}$ such that $\mathit{type}(\mathit{rmsg}_{\sigma(c)}) = \langle \sigma(s_1), \ldots, \sigma(s_n)\rangle$, and $\mathit{syn}_{\sigma(c)} \in \mathcal{A}_2$ such that $\mathit{type}(\mathit{syn}_{\sigma(c)}) = \langle \mathsf{addr}, \sigma(s_1), \ldots \sigma(s_n), \mathsf{bool}\rangle \to \mathsf{bool}$

4. For each $c \in \Gamma_{(e_1 - e_{b_1}) \cap (l_1 - l_{b_1})}$, $\mathit{rep}_{\sigma(c)} = \mathit{ack}_{\sigma(c)}$.

The set $\Psi_2$ is determined by $\sigma(\Psi_2)$ and the three families of axioms below. The first of these defines the behaviour of actors playing the sender role in the protocol description above:

**(S1)** $\bigwedge_{c \in \sigma(\Gamma_{e_1 - e_{b_1}})} \forall \vec{v_c} \cdot n.\mathbf{init} \to n.\mathit{wait}_c(m, \vec{v_c}) = \mathrm{F}$

**(S2)** $\bigwedge_{c\in\sigma(\Gamma_{e_1-e_{b_1}})} \forall\vec{v_c}\cdot n.switch_c(k,\vec{v_c}) \wedge n.wait_c(k,\vec{v_c}) = x \rightarrow \mathbf{X}(n.wait_c(k,\vec{v_c}) = \textsc{not}(x))$

**(S3)** $\bigwedge_{c\in\sigma(\Gamma_{e_1-e_{b_1}})} \forall\vec{v_c}\cdot \neg n.switch_c(k,\vec{v_c}) \wedge n.wait_c(k,\vec{v_c}) = x \rightarrow \mathbf{X}(n.wait_c(k,\vec{v_c}) = x)$

**(S4)** $\bigwedge_{c\in\sigma(\Gamma_{e_1-e_{b_1}})} \forall\vec{v_c}\cdot (n.\mathbf{send}\ c,k,\vec{v_c}\ (\vee) n.ack_c(k,\vec{v_c})) \wedge k \neq n \rightarrow \mathbf{X}(n.switch_c(k,\vec{v_c}))$

**(S5)** $\bigwedge_{c\in\sigma(\Gamma_{e_1-e_{b_1}})} \forall\vec{v_c}\cdot n.switch_c(k,\vec{v_c}) \leftarrow (n.\mathbf{send}\ c,k,\vec{v_c}\ (\vee) n.ack_c(k,\vec{v_c})) \wedge k \neq n$

**(S6)** $\bigwedge_{\substack{c\in\sigma(\Gamma_{e_1-e_{b_1}})\\ d\in\sigma(\Gamma_{l_1-l_{b_1}})}} \forall\vec{v_c},\vec{v_d}\cdot n.d(\vec{v_d}) \vee n.\mathbf{deliv}\ (\vec{v_d}) \rightarrow n.wait_c(k,\vec{v_c}) = \mathrm{F}$

**(S7)** $\bigwedge_{c\in\sigma(\Gamma_{e_1-e_{b_1}})} \forall\vec{v_c}\cdot n.wait_c(k,\vec{v_c}) = \mathrm{T} \rightarrow \mathbf{FE}(n.\mathbf{deliv}\ (ack_c,k,\vec{v_c})) \wedge \mathbf{FE}(n.ack_c(k,\vec{v_c}))$

The second family of axioms specifies the behaviour of actors playing the receiver role, which is considerably more complex than the previous one:

**(R1)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.\mathbf{init} \rightarrow n.syn_c(k,\vec{v_c},x) = \mathrm{F}$

**(R2)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.rmsg_c(\vec{v_c}) \wedge n.syn_c(k,\vec{v_c},\mathrm{T}) = x \rightarrow \mathbf{X}(n.syn_c(k,\vec{v_c},\mathrm{T}) = \textsc{not}(x))$

**(R3)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.rid_c(k,\vec{v_c}) \wedge n.syn_c(k,\vec{v_c},\mathrm{F}) = x \rightarrow \mathbf{X}(n.syn_c(k,\vec{v_c},\mathrm{F}) = \textsc{not}(x))$

**(R4)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot \neg n.rid_c(k,\vec{v_c}) \wedge \neg n.rmsg_c(\vec{v_c}) \wedge n.syn_c(k,\vec{v_c},x) = y \rightarrow \mathbf{X}(n.syn_c(k,\vec{v_c},x) = y)$

**(R5)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.c(\vec{v_c}) \wedge \nexists k\cdot n.syn_c(k,\vec{v_c},\mathrm{F}) = \mathrm{T} \rightarrow \mathbf{X}(n.rmsg_c(\vec{v_c}))$

**(R6)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.c(\vec{v_c}) \wedge n.syn_c(k,\vec{v_c},\mathrm{F}) = \mathrm{T} \rightarrow \mathbf{X}(n.\mathbf{send}\ rep_c,k,\vec{v_c}\ (\wedge) n.rid_c(k,\vec{v_c}))$

**(R7)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.id_c(k,\vec{v_c}) \wedge n.syn_c(k,\vec{v_c},\mathrm{F}) = \mathrm{T} \rightarrow \mathbf{X}(n.rid_c(k,\vec{v_c}))$

**(R8)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.id_c(k,\vec{v_c}) \wedge n.syn_c(k,\vec{v_c},\mathrm{T}) = \mathrm{T} \rightarrow \mathbf{X}(n.\mathbf{send}\ rep_c,k,\vec{v_c}\ (\wedge) n.rmsg_c(\vec{v_c}))$

**(R9)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.\mathbf{send}\ rep_c,k,\vec{v_c}\ () \leftarrow n.id_c(k,\vec{v_c}) \wedge n.syn_c(k,\vec{v_c},\mathrm{T}) = \mathrm{T} \vee n.c(\vec{v_c}) \wedge n.syn_c(k,\vec{v_c},\mathrm{F}) = \mathrm{T}$

**(R10)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.rmsg_c(\vec{v_c}) \leftarrow n.id_c(k,\vec{v_c}) \wedge n.syn_c(k,\vec{v_c},\mathrm{T}) = \mathrm{T} \vee n.c(\vec{v_c}) \wedge \nexists k\cdot n.syn_c(k,\vec{v_c},\mathrm{F}) = \mathrm{T}$

**(R11)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.rid_c(k,\vec{v_c}) \wedge k \neq n \leftarrow n.id_c(k,\vec{v_c}) \wedge n.syn_c(k,\vec{v_c},\mathrm{T}) = \mathrm{F} \vee n.c(\vec{v_c}) \wedge n.syn_c(k,\vec{v_c},\mathrm{F}) = \mathrm{T}$

**(R12)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.id_c(k,\vec{v_c}) \vee n.\mathbf{deliv}\ (id_c,k,\vec{v_c}) \rightarrow n.sync_c(k,\vec{v_c},\mathrm{F}) = \mathrm{F}$

**(R13)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.c(\vec{v_c}) \vee n.\mathbf{deliv}\ (c,\vec{v_c}) \rightarrow n.sync_c(k,\vec{v_c},\mathrm{T}) = \mathrm{F}$

**(R14)** $\bigwedge_{c\in\sigma(\Gamma_{l_1-l_{b_1}})} \forall\vec{v_c}\cdot n.syn_c(k,\vec{v_c},\mathrm{F}) = \mathrm{F} \rightarrow \mathbf{FE}(n.\mathbf{deliv}\ (id_c,k,\vec{v_c})) \wedge \mathbf{FE}(n.id_c(k,\vec{v_c}))$

Finally, we also need to propose a family of axioms defining the additional behaviour of actors playing both roles in the same interaction, i.e., those actors self-addressing a message:

**(SR1)** $\bigwedge_{c\in\sigma(\Gamma_{(e_1-e_{b_1})\cap(l_1-l_{b_1})})} n.\mathbf{send}\ c, n, \vec{v_c}\ (\rightarrow)\mathbf{F}(n.id_c(n, \vec{v_c}))$

**(SR2)** $\bigwedge_{c\in\sigma(\Gamma_{(e_1-e_{b_1})\cap(l_1-l_{b_1})})} n.\mathbf{send}\ c, n, \vec{v_c}\ (\rightarrow)(\neg n.\mathbf{deliv}\ (c, \vec{v_c}))\mathbf{W}(n.id_c(n, \vec{v_c}))$

**(SR3)** $\bigwedge_{c\in\sigma(\Gamma_{(e_1-e_{b_1})\cap(l_1-l_{b_1})})} \forall\vec{v_c}\cdot n.\mathbf{send}\ c, n, \vec{v_c}\ (\leftarrow)n.syn_c(n, \vec{v_c}, \mathrm{F}) = \mathrm{F}$ $\square$

Let us clarify the meaning of the first set of axioms. **S1** asserts that the originator is not initially in await state, i.e. blocked. **S2** and **S3** say that this state is only reached and abandoned due to the occurrence of the local computation $switch_c$. This computation is caused and only happens due to the dispatch of the message $c$ or the acknowledgement of its receipt, according to **S4** and **S5**. Note that self-addressed messages are excluded from this causality relationship. **S6** specifies that the wait state consists in forbidding the delivery and processing of base specification messages. Finally, **S7** asserts that it is eventually possible to receive and process a message receipt acknowledgement in a wait state.

The second set of axioms is similar to the previous one, but addresses more complex situations faced by actors performing the recipient role. Recipients are not initially in await state, which is reached if either the contents of a message or the sender address are processed by local computations, which happen only because of the processing of the respective messages. This is what **R1-8** determine. **R9-11** define the precedence conditions which hold about the occurrence of local computations and the dispatch of replies. Note that we leave untreated local computations dealing with the identification of self-addressed messages because in this case the recipient would also be playing the originator role and this is to be treated by the subsequent set of axioms. **R12** says that await states are not strict, preventing only the delivery and consumption of dispatched message identities with contents identical to previously consumed but not identified ones. **R13** states the same regarding the delivery and consumption of message contents. **R14** is the usual enabledness axiom concerning *id*.

To complete the specification of the synchrony protocol, we propose the third set of axioms above. **SR1** says that the dispatch of self-addressed messages implies that they are eventually self-identified using the respective local computation. Additionally, **SR2** prevents the delivery of self-addressed messages from happening before self-identification. As a result of these axioms, we have that whatever causes the dispatch of a self-addressed message also causes a self-identification. The remaining axiom, **SR3**, requires that self-addressed messages be dispatched only if previously dispatched identical messages are to be delivered and processed first. With this special treatment of self-addressed messages, we prevent the components complying with the resulting specifications from deadlocking strictly as a result of the transformation[4]. It is also important to mention that, due to the same treatment, the specification of the transformation in the form of a morphism has to be stated at the global level. This happens because the specified extended behaviour considers that it is necessary for each object to have knowledge about its own name. Otherwise, it would not be possible to expect the acknowledgement of the receipt of each message. As usually present in many object-based programming languages in the form of a built-in object variable, the existence of a $self$ attribute symbol in our formalism would allow us to propose a local definition for the transformation above.

Note that, by formalising the transformation in the way above, we obtain a method which is not fully compositional in that, if applied to each member in isolation originating a co-limit diagram, i.e., to each given specification, the connection of the resulting objects by a co-limit of specification morphisms would not automatically correspond to the application of the same transformation to the co-limit object of the whole original diagram. This happens because some message symbols which have to be equalised by the transformation could remain untreated, namely those related by $\sim$ in Figure 3.5. To confirm this in practice, the reader is invited to apply the transformation to COMPONENT1, TERMINAL and BUFFERCELL, and to observe that not every pushout of the transformation of CONNECTOR1 along arbitrary morphisms into the latter two specifications after transformation equalises $ack_{tr}$ and $rep_{put}$, $orig_{tr}$ and $id_{put}$. This only happens using morphisms that conform with the translation of the base specification symbols and translate the auxiliary ones accordingly.

Some other subtleties result from the application of the transformation above. For instance, safety properties are not necessarily "preserved". The typical example is the emergence of deadlocks (Charron-Bost *et al.* 1996). These may appear not only due to a misleading definition (which we have striven to avoid in Definition 3.7.1), but also because of the specific properties captured by the original specification. It must also be clear that, if some objects satisfy the requirements of a transformed specification, they cannot freely interact with an environment which does not obey the same requirements. For example, a message which is received from an environment which never bothers to identify the sender eventually leads to a deadlock. In other words, the transformation reduces openness. This is not surprising: in methods wherein each message is augmented with protocol dependent information — see (Sturman and Agha 1995) for an example — an object that does not comply with the protocol may not have sufficient knowledge to deal with extended messages.

[4]Note that these components may deadlock anyway while adopting a synchronous mode of interaction due to their inherent properties.

Despite the limitations of our method enumerated above, it is important to recall that it is indeed possible to represent synchronous and other less stringent modes of interaction in terms of the actor model. An example of a synchronous system is obtained by applying the transformation above to the specification UTSA and by considering the resulting system as described in Section 3.6.

## 3.8 Actors and Dynamic Subclassing

It has become customary to consider the notion of *class* in object-based design. Classes are collections of objects which obey the same definition, be it a program or a specification[5]. They are normally coupled not only to a method which permits easy reuse of definition parts, *inheritance*, but also to a relation between properties of class elements, *subtyping*. Due to the inclusion relation between sets, a *sub-class* relation is readily induced. These notions are not assumed in the definition of the actor model, according to Wegner (1987); so, they can be easily superimposed to produce a particularised model. Taking specifications into account, here we may consider that actor communities are classes, the respective specification morphisms determine relations of (possibly multiple) inheritance and the induced notion of theory inclusion characterises subtyping.

The introduction of the notions above in the actor model does not appear to be interesting *per se*. Nevertheless, it can lead to an elegant treatment of extensibility other than just by means of object creation and reconfiguration. Many authors including Wieringa *et al.* (1995) have studied dynamic notions of class wherein objects are allowed to migrate from a class to another at run time. This is called *dynamic sub-classing*. In our example concerning buffer cells, it would be possible in this way to allow cells in the Full class to become Empty. Formally, Wieringa *et al.* (1995) consider that a non-trivial dynamic partition of a class is a collection of sets of class elements such that their union is equal to the whole class, these sets are pairwise disjoint and, in addition, there is some behaviour in which an object goes from one set to another, for each two such sets in the dynamic class partition. A *dynamic subclass*, in turn, is a set in a dynamic partition of a class. By allowing an object to migrate between dynamic subclasses, it is possible to support extension (and restriction) of functionality.

In this context, let us look at our previous example in more detail. Figure 3.6 illustrates how buffer cells can be organised taking into account the notion of dynamic subclass. There are two ways of dynamically partitioning such a class, not only according to the empty or full character of cells but also considering

[5]Here we need to clarify that this is just one of the many possible definitions of class available in the literature

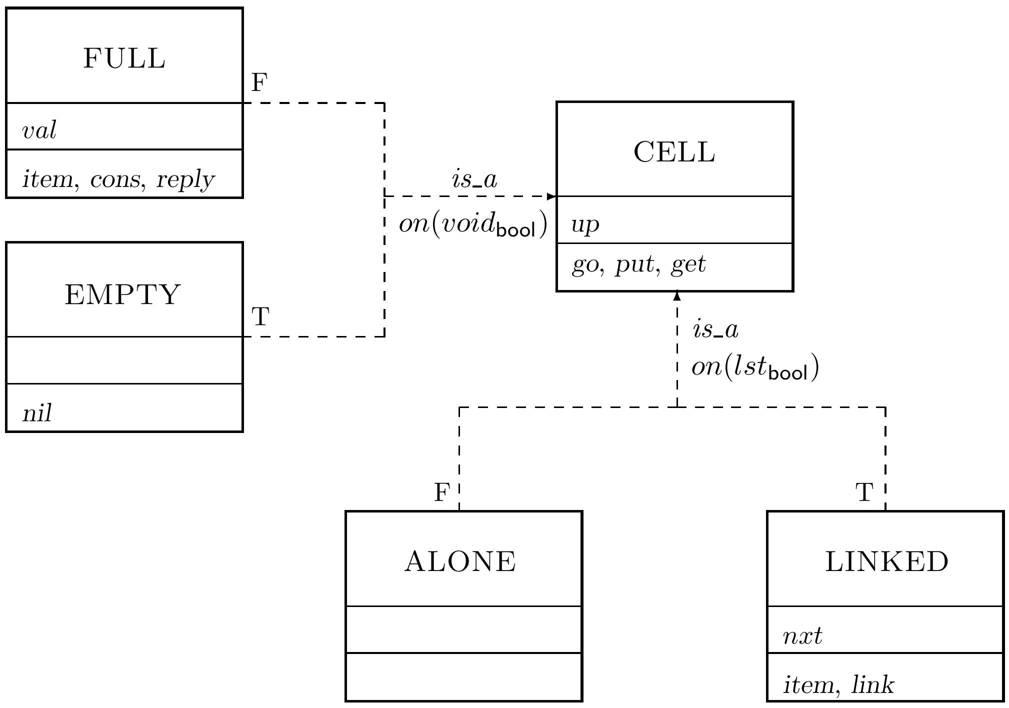


Figure 3.6: Buffer cells and related dynamic subclasses.

that they may or may not be logically linked to other similar objects. In the picture, we represent the class and its respective subclasses using boxes, which are divided in three regions to allow the representation of the name, attribute and action symbols of the class. The relation of being a member of the same dynamic partition of a class is represented using connected dashed lines, which are joined together to express the fact that a set of dynamic subclasses is being defined. Similar diagrams are usual in object-oriented design.

We adopt some auxiliary notation in this kind of informal software diagram to express how an object is identified as a member of a dynamic subclass. To this effect, Wieringa *et al.* (1995) use both logical class predicates and retracts in more detailed specifications. Here, because we prefer to use our distinct underlying formalism, we choose instead dynamic subclass selectors, which must be provided as part of each superclass specification. We use *on* as a keyword in each diagram to say which attribute plays this role. Note that each value determining subclass membership is written as a diagram annotation in the picture and this set of values must be in a one to one correspondence with the dynamic classes in the partition. Some actions in the respective subclasses are also needed to capture the events of subclass migration.

In our example, *void* distinguishes full from empty cells. The action *cons*

$$\begin{tikzcd}
 & & \textsc{BufferCell} & & \\
\textsc{Alone} \arrow[urr] & \textsc{Empty} \arrow[ur] & & \textsc{Full} \arrow[ul] & \textsc{Linked} \arrow[ull] \\
 & \textsc{Cell} \arrow[ul, "\iota_1"] \arrow[u, "\iota_2"] \arrow[urr, "\iota_3"] \arrow[urrr, "\iota_4"'] & & &
\end{tikzcd}$$

Figure 3.7: Static configuration of the dynamic subclasses of Cell.

is responsible for the migration of objects from one class to the other. The pair consisting of *lst* and *link* plays the same role with respect to the partition of cells into those which are linked or alone. In this situation, we must also treat the birth of objects in the respective subclasses. Wieringa *et al.* (1995) remark that only *species*, smallest classes partitioning the universe such as Full $\cap$ Linked, should be assigned to creation events. In our example, such events correspond to the occurrence of either *item* or *nil* depending on whether or not the cell is to be full or empty. Cells are created isolated and thus the prescription to introduce creation events only in species is fulfilled. The use of birth action symbols to represent object creation uncovers an important issue: that objects of diverse sub-classes may need to use the same symbol to request a birth. Since each class has a separate specification, this can only be treated by requiring the existence of morphisms to identify these symbols as representing the same event. For instance, a pair of morphisms can make explicit that the action *item* of Linked is the same as in the Full class. As a result, we obtain that informal diagrams as in Figure 3.6 resemble the structure of the categorical diagram with reversed arrows that could be used to describe the same situation. A co-limit diagram describing this class structure can be organised as in Figure 3.7.

The formal diagram in the aforementioned figure does not make such sense without a definition of the related specifications. These are presented in Figure 3.8. We consider that the involved morphisms are all identities. The axioms in those specifications correspond to the properties of BufferCell, which turned out to be captured in separated sets of axioms by the approach based on dynamic sub-classing. Class selectors, birth and migration actions are all included in the superclass specification. Each sub-class specification only constrains the

**Specification** $\textsc{Cell}$
**data types** $\mathsf{addr}, \mathsf{bool}, \mathsf{int}$ $(\mathrm{T}, \mathrm{F} : \mathsf{bool})$
**attributes** $\mathit{void}, \mathit{lst}, \mathit{up} : \mathsf{bool}$
**actions** $\mathit{nil}, \mathit{item}(\mathsf{int})$ : **local** + **extrn birth**;
$\mathit{go}, \mathit{const}, \mathit{link}(\mathsf{addr})$ : **local computation**;
$\mathit{put}(\mathsf{int}), \mathit{get}(\mathsf{addr})$ : **local** + **extrn message**;
**axioms** $k, n : \mathsf{addr}; v : \mathsf{int}; x, y : \mathsf{bool}$
$\mathit{go} \wedge \mathit{void} = x \wedge \mathit{lst} = y \rightarrow \mathbf{X}(\mathit{up} = \mathrm{T} \wedge \mathit{void} = x \wedge \mathit{lst} = y)$ (14.1)
$\mathit{cons} \wedge \mathit{lst} = x \wedge \mathit{up} = y \rightarrow \mathbf{X}(\mathit{void} = \mathrm{T} \wedge \mathit{lst} = x \wedge \mathit{up} = y)$ (14.2)
$\mathit{link}(n) \wedge \mathit{void} = x \wedge \mathit{up} = y \rightarrow \mathbf{X}(\mathit{lst} = \mathrm{F} \wedge \mathit{void} = x \wedge \mathit{up} = y)$ (14.3)
$\mathit{up} = \mathrm{T} \rightarrow \mathbf{FE}(\mathbf{deliv}\ (\mathit{put}, v)) \wedge \mathbf{FE}(\mathit{put}(v)) \wedge \mathbf{FE}(\mathbf{deliv}\ (\mathit{get}, n)) \wedge \mathbf{FE}(\mathit{get}(n))$ (14.4)
**End**

**Specification** $\textsc{Empty} = \iota_2(\textsc{Cell}) +$
**axioms**
$\mathit{nil} \rightarrow \mathit{void} = \mathrm{T} \wedge \mathit{lst} = \mathrm{T} \wedge \mathit{up} = \mathrm{F}$ (15.1)
$\mathit{nil} \rightarrow \mathbf{X}(\mathit{go})$ (15.2)
**End**

**Specification** $\textsc{Full} = \iota_3(\textsc{Cell}) +$
**attributes** $\mathit{val} : \mathsf{int}$;
**actions** $\mathit{reply}(\mathsf{int})$ : **extrn message**
**axioms** $n : \mathsf{addr}, v : \mathsf{int}$
$\mathit{item}(v) \rightarrow \mathit{val} = v \wedge \mathit{void} = \mathrm{F} \wedge \mathit{lst} = \mathrm{T} \wedge \mathit{up} = \mathrm{F}$ (16.1)
$\mathit{item}(v) \rightarrow \mathbf{X}(\mathit{go})$ (16.2)
$\mathit{go} \wedge \mathit{val} = v \rightarrow \mathbf{X}(\mathit{val} = v)$ (16.3)
$\mathit{get}(n) \wedge \mathit{void} = \mathrm{F} \wedge \mathit{val} = v \rightarrow \mathbf{X}(\mathbf{send}\ \mathit{reply}, n, v\ (\wedge)\mathit{cons})$ (16.4)
$\mathbf{send}\ \mathit{reply}, n, v\ (\vee)\mathit{cons} \leftarrow \mathit{get}(n) \wedge \mathit{val} = v \wedge \mathit{void} = \mathrm{F}$ (16.5)
**End**

**Specification** $\textsc{Linked} = \iota_4(\textsc{Cell}) +$
**attributes** $\mathit{nxt} : \mathsf{addr}$;
**axioms** $k, n : \mathsf{addr}; v : \mathsf{int}$
$\mathit{link}(n) \rightarrow \mathbf{X}(\mathit{nxt} = n)$ (17.1)
$\mathit{go} \wedge \mathit{nxt} = n \rightarrow \mathbf{X}(\mathit{nxt} = n)$ (17.2)
$\mathit{put}(v) \wedge \mathit{lst} = \mathrm{F} \wedge \mathit{nxt} = n \rightarrow \mathbf{X}(\mathbf{send}\ \mathit{put}, n, v\ ())$ (17.3)
$\mathit{get}(n) \wedge \mathit{void} = \mathrm{T} \wedge \mathit{lst} = \mathrm{F} \wedge \mathit{nxt} = k \rightarrow \mathbf{X}(\mathbf{send}\ \mathit{get}, k, n\ ())$ (17.4)
$\mathit{put}(v) \wedge \mathit{lst} = \mathrm{T} \rightarrow \mathbf{X}(\exists n \cdot \mathbf{new}(\mathit{item}, n, v) \wedge \mathit{link}(n))$ (17.5)
$\mathbf{send}\ \mathit{put}, k, v\ (\leftarrow)\mathit{put}(v) \wedge \mathit{nxt} = k \wedge \mathit{lst} = \mathrm{F}$ (17.6)
$\mathbf{send}\ \mathit{get}, k, n\ (\leftarrow)\mathit{get}(n) \wedge \mathit{nxt} = k \wedge \mathit{void} = \mathrm{T} \wedge \mathit{lst} = \mathrm{F}$ (17.7)
$\exists n \cdot \mathbf{new}(\mathit{item}, n, v) \vee \mathit{link}(n) \leftarrow \mathit{put}(v) \wedge \mathit{lst} = \mathrm{T}$ (17.8)
**End**

Figure 3.8: Specification of the distinct dynamic subclasses of $\textsc{Cell}$.

properties of the symbols that appear inside the respective boxes in Figure 3.6. We are obliged to choose this structure for specifications of the class hierarchy by the logical properties of our underlying model. It is important to emphasise that because of the locality property, an object in a specific dynamic class may only change the object attributes specified in the respective class description. Since dynamic subclasses must not be defined at run time but at the time of system description, we see that the advantage dynamic sub-classing appears to offer is the modularisation of component descriptions. Since the actor model can already deal with the notions of state and change, we can simply capture this other dynamic notion by adopting a specific design discipline.

## 3.9 Summary and Related Work

In this chapter, we have particularised the logical system previously proposed in order to support the design of open reconfigurable systems in a more faithful way. We chose to provide built-in support for the actor model, which captures both openness and reconfigurability. The structure of each signature was specialised to cater for the finer distinctions between the families of symbols present in each actor specification. A set of logical axioms was proposed to capture such distinctions in meaning and to pose additional constraints in the specified object behaviour. We also defined a syntactic way of composing actor specifications through the same categorical constructions studied in the previous chapter. A rely-guarantee discipline supporting the verification of dynamic properties was established. An example was used to illustrate local and global reasoning.

The use of additional logical symbols to represent complex object behaviour is not new. Ehrich *et al.* (1988) introduced the idea of adding a new argument in each signature symbol to represent object identity. Wieringa *et al.* (1995) used additional flexible symbols to represent classes and object existence. As an alternative to both techniques, we could have adopted sort symbols with a flexible meaning, at the expense of using a substantially more complex underlying temporal logical system. In any case, the introduction of such logical symbols and the use of a set of abbreviations appears to be the best choice when we consider that unique identification of objects and messages has to be supported without loosing our intuitions about the actor model.

In the literature on actors, we can find plenty of examples on rigorous approaches to the model. Talcott (1996a) provides an operational semantics for actors defined in terms of the application of rewriting logic rules. The inference rules of linear logic play the same role in the work of Darlington and Guo (1995).

The detailed operational semantics in (Agha *et al.* 1997) is defined in terms of a transition relation on actor configurations. All these works appear to deal with the semantics of actor programs only. The early studies of Hewitt and Baker (1977) and of Clinger (1981) were entirely semantic. So, our work seems to be the first to deal with the formal design of open reconfigurable systems based on this model. The means to support actor specification and verification appear to be the main contribution of this chapter.

Openness and reconfigurability have been addressed in the recent literature, receiving special attention from those who advocate an object-based approach to software design. Fiadeiro and Maibaum (1992) and also Sernadas *et al.* (1995) capture openness in a static object configuration setting considering that each specified event may occur in parallel to other events of the environment. This semantics for action symbols was adopted here as well. America and de Boer (1996) develop an extensive study of dynamically reconfigurable synchronous object communities and provide methods for reasoning about their properties. To support the proof of global properties, in particular, a cooperation test written at the global level has to be proved. Here, on the contrary, because interaction is always asynchronous, the decision as to when to accept a message is purely local. Abadi and Leino (1997) propose a Hoare logic of object-oriented programs. Note that Hoare logics are usually endowed with a set of inference rules supporting the verification of general conclusions from particular assertional premises, which are solely based on the state of the system in a single (pair of) instant(s). Here, we have adopted a distinct strategy with our derived inference rules, which prioritises instead the separation of properties pertaining to the distinct objects involved in each interaction. This appears to facilitate the development of proofs taking only into account their possibly separated specifications, thus reducing the proof search space.

A number of methods supporting a rely-guarantee discipline has already appeared in the literature with the aim of supporting the design of open systems. Pandya and Joseph (1991) develop in the realm of synchronous process calculi a theory which appears to be the closest to our work. Other related work can be roughly divided in process or model based formalisms (Jones 1983, Cau and Collete 1996) and logical ones (Pnueli 1985b, Collete 1994, Abadi and Lamport 1995, Jonsson and Tsay 1995). Unfortunately, in the latter recent work, many distinct levels of abstraction are discussed without a clear boundary, due to the influential view that implication coincides with refinement, as advocated by Abadi and Lamport (1995). In this latter category, only two kinds of assertions representing assumptions and commitments are considered. All these works allow the use of arbitrary safety properties but just a few consider the occurrence of liveness properties as a normal part of commitment assertions. In our work, both families of properties are treated uniformly as any part of rely-guarantee assertions. In particular, the use of the connectives unless and until to relate past and future relieve us from adopting the more demanding semantic closures and history variables in the application of composition rules. On the other hand, we have not studied in detail yet how to treat hidden flexible variables.

In order to provide evidence that there is no loss of expressiveness by adopting the asynchronous actor model in the design of open reconfigurable systems, we exemplified how object descriptions can be transformed into constrained specifications which force the behaviour of each system to comply with a synchronous mode of interaction. Agha *et al.* (1994) also discuss a number of higher-level abstractions defined in terms of the actor model, including the treatment of less asynchronous interaction modes. In particular, synchronisation constraints are treated, which permit the receipt of a message to be delayed until the object is in a state where it is possible to proceed with the processing of the message. Note that this is specified here in a way similar to (13.11), by relating the possible delivery or consumption of a message using our modal possibility connective to the local state of the object and the message contents. There is a clear advantage in using such constraints in relation to a synchronous mode of interaction, namely that they do not block the sender. The same applies to the call/return abstraction described in that work. Note that there is a fundamental distinction between these abstractions and the use of our synchrony transformation: they are to be used by designers and programmers as part of actor behaviour descriptions whereas transformations of the kind studied here consider that such descriptions have already been produced. Further work related to such transformations is proposed in Chapter 6.

We also made a digression concerning the use of dynamic subclasses as an object-oriented approach to support extensibility. We showed that, because the actor model can capture state and change, dynamic subclasses can also be designed in terms of this model by a specific design discipline which leads to more modular specifications. Because change is normally causally connected to the occurrence of interaction here, which in turn usually eventually happen due to the format of the axioms necessary in each actor specification, we obtain a design notion which complies not only with the definition of dynamic subclass but also with our definition of extensibility in Chapter 1. Our work differs in a few points from that described by Wieringa *et al.* (1995). First of all, interaction is not discussed therein. Also, because our underlying model supports unique object identification, this does not need to be treated in the study of dynamic subclasses. Finally, we do not require that class migration be irreflexive: here an object may migrate to the same class it currently belongs to. In this way, class migration coincides with the intended meaning of the actor primitive **become**.

# Chapter 4

# Reflection and the Design of Meta-Level Architectures

The open reconfigurable system abstraction is useful to support the design of software systems *in the small.* Recently, however, the trend has been to pay more and more attention to the overall organisation of the components of each system and their interrelationships, classifying the distinct ways in which they are designed, organised and evolve over time with the aim of providing automated development tools and supporting reuse. The branch of Software Engineering concerned with these issues is called *software architecture* (Garlan 1995).

Conventional architectural styles have been identified in existing systems and have guided new designs. Examples are the client-server and pipe-and-filter styles. The most non-conventional style is perhaps that of meta-level architectures. A *meta-level architecture* is one wherein there is a clear separation of components into *base-level objects*, which are devoted to solving a problem in the application domain, and *meta-level objects*, which deal with the base-level of the architecture itself — its configuration, operational behaviour and the way it is used to accomplish the main purpose of the system. This separation may be iterated to identify many (possibly unrelated) meta-levels in the same architecture. Meta-level objects are useful in applications such as memory management, debugging, fault detection and recovery. Particularly in the context of concurrent and distributed systems, this separation is important in such activities as scheduling, load balancing and task migration. In this way, meta-level objects do not directly deal with the problem domain, but help in establishing a better organisation of the system, as observed by Simhi *et al.* (1996).

When the meta relation is recursively iterated, architectures complying with a distinguished style are obtained. *Reflective architectures* realise computational reflection. Maes (1987) characterises *computational reflection* as the activity performed by each component when doing computation about its self, possibly affecting its own behaviour. In a reflective system, the meta-levels are represented by an interpreter and there is a *causal connection* between system description and its behaviour: whenever the description changes, the behaviour changes as a result and each modification in behaviour is preceded by a description change. Programming and specification languages are said to be *reflective* if reflection is explicitly supported by specific language constructs. In the former case, such languages are said to have an underlying reflective architecture.

Many authors have studied the design of meta-level and reflective architectures. Simhi *et al.* (1996) propose a technique based on state transition diagrams to enhance the design of reflective objects. Tahara *et al.* (1996) introduce an algebraic semantics for reflective objects based on an extension of *rewriting logic* (Meseguer 1992). Saeki *et al.* (1993) propose a reflective extension of the specification language LOTOS. Clavel and Meseguer (1996) study reflection in a general logical setting and show that the executable specification language Maude, which is based on rewriting logic, fulfils the conditions to be reflective.

It is important to stress that the research mentioned above is mostly concerned with the design of meta-level and reflective architectures. In logic, meta-theoretic facilities have also been studied without any required connection with a notion of computation. Such theoretic study involves axiomatising a given provability relation and this allows one to use the logical language to talk about the logical system itself. This is why such facilities are said to be *introspective.* Attardi and Simi (1991), Basin and Matthews (1996) pursue this line of research.

The confusion between the presence of meta-level or reflective facilities at the architectural level and in the logical system used for design is just one of the many points that has remained rather unclear concerning these notions. Most authors do not distinguish meta-level from reflective architectures as Venkatasubramanian and Talcott (1993) do. Moreover, it is not clear in many situations if these are required or even desirable notions. On the other hand, both notions are clearly helpful in ensuring extensibility due to the possibility of providing extended base-level functionality as a result of meta-level behaviour.

In this chapter, we first show that the assumption of meta-level architectures is reasonable in the design of open reconfigurable systems as formalised in the previous chapter. We base our rationale on the impossibility of solving the consensus problem in asynchronous systems that admit crash failures (Fischer *et al.* 1985). Next, we define a discipline that permits the design of meta-level architectures. Finally, we argue that the assumption of reflective architectures is not compatible with systematic software development due to the impossibility of relying on constantly changing specifications for verification and refinement.

## 4.1 Meta-level Considered Necessary: The Consensus Problem

In this section, we show that the assumption of meta-level architectures is reasonable in the design of open reconfigurable systems as formalised in the previous chapter. To reach this conclusion, we use the *consensus problem*, which involves a set of processes which may individually fail but have to agree on the same binary value otherwise. This is just an abstraction of many practical problems such as distributed transaction commitment. Fischer *et al.* (1985) show that it is impossible to find an implementation that solves this problem in a completely asynchronous setting admitting at least one unreliable process.

Many distinct types of failure are studied in the design of fault-tolerant systems. *Message loss* is the most typical example in a message passing mode of interaction. The consensus problem is impossible in the presence of *crash-failures* (also known as *fail-stop failures*) or more severe ones such as *Byzantine failures*, which may be followed by an arbitrary object behaviour, even in an ideal reliable network that guarantees message delivery. This last property is ensured by our axiomatisation of the actor model. On the other hand, due to our decision to make weaker assumptions than those of perfect message buffering in our axiomatisation, here it is possible to represent crash-failures as required in any attempt to deal with agreement problems.

The many processes of a system involved in reaching distributed agreement are assumed to hold an initial boolean value and to interact solely by asynchronous message passing. For any such an agreement system to be correct, the following properties are required to hold:

**Termination:** Every non-faulty process eventually decides some value;

**Agreement:** Each pair of non-faulty processes decides the same value;

**Integrity:** Every process decides at most once;

**Validity:** If a process decides a value, it was the initial value of some process.

Note that integrity is local whereas the other ones are global properties of the system. Weaker formulations of the consensus problem also yield an impossibility result, but for our illustrative purposes the formulation above suffices.

Due to the general nature of the problem, it appears to be more profitable to attempt a general treatment here. In order to represent agreement processes, we use presentation schemas, which differ from theory presentations just because the signature symbols are left partially unspecified and the stated

**Specification** UNRELPROC
**data types** $\Omega \cup \{\mathsf{addr}, \mathsf{bool}\ (\mathrm{T}, \mathrm{F} : \mathsf{bool})\}$
**attributes** $\mathcal{A} \cup \{failed : \mathsf{bool}\}$
**actions** $\Gamma_{l_b}$ : **local birth**;
$\Gamma_{e_b}$ : **extrn birth**;
$\Gamma_c \cup \{\mathit{fail}\}$ : **local computation**;
$\Gamma_{l-l_b}$ : **local message**;
$\Gamma_{e-e_b}$ : **extrn message**
**axioms** $x : \mathsf{bool}, n : \mathsf{addr}$

$$\mathit{fail} \rightarrow failed = \mathrm{F} \tag{18.1}$$

$$\mathit{fail} \rightarrow \mathbf{X}(failed = \mathrm{T}) \tag{18.2}$$

$$\bigwedge_{c \in \Gamma_c} \exists \vec{v_c} \cdot c(\vec{v_c}) \wedge failed = x \rightarrow \mathbf{X}(failed = x) \tag{18.3}$$

$$\bigwedge_{b \in \Gamma_{e_b}} \exists n, \vec{v_b} \cdot \mathbf{new}(b, n, \vec{v_b}) \rightarrow failed = \mathrm{F} \tag{18.4}$$

$$\bigwedge_{b \in \Gamma_{l_b}} \exists \vec{v_b} \cdot b(\vec{v_b}) \rightarrow failed = \mathrm{F} \tag{18.5}$$

$$\bigwedge_{c \in \Gamma_c} \exists \vec{v_c} \cdot c(\vec{v_c}) \rightarrow failed = \mathrm{F} \tag{18.6}$$

$$\bigwedge_{c \in \Gamma_{l-l_b}} \exists \vec{v_c} \cdot \mathbf{deliv}\ (c, \vec{v_c}) \vee c(\vec{v_c}) \rightarrow failed = \mathrm{F} \tag{18.7}$$

$$\bigwedge_{c \in \Gamma_{e-e_b}} \exists n, \vec{v_c} \cdot \mathbf{send}\ c, n, \vec{v_c}\ (\rightarrow) failed = \mathrm{F} \tag{18.8}$$

**End**

Figure 4.1: Schematic specification of unreliable processes.

sentences may be schemas and not just axioms as usual. Whenever we refer to one such schematic presentation, we are in fact making reference to all the theory presentations which have a signature and a set of axioms complying with the specified syntactic pattern. In addition, each morphism connecting a source to a target schematic presentation is assumed to represent a family of morphisms relating the respective theory presentations which also respect the translation of the source axiom schemas.

Our schematic presentation of unreliable processes appears in Figure 4.1. Each process is endowed with distinguished local computation and boolean attribute symbols, *fail* and $failed$ respectively, which represent the occurrence of a failure and the unreliable state reached as a result of this occurrence. Axiom (18.1) says that failures can occur only in reliable states. Moreover, an unreliable state is reached as a result of such an occurrence and only then, according to (18.2) and (18.3). Schema (18.4) specifies that only initially reliable processes are admissible. The other axiom schemas in the presentation determine that it is impossible for the process to witness the occurrence of local events after the

**Specification** $\textsc{UnrelAgmProc} = \iota(\textsc{UnrelProc}) +$
**attributes** $\mathit{initial}, \mathit{decision}, \mathit{decided} : \mathsf{bool}; \mathit{known} : \mathsf{addr}^n \ (n \in \mathbf{N})$
**actions** $\mathit{decide}(\mathsf{bool})$ : **local computation**
**axioms** $x, y : \mathsf{bool}; \vec{k} : \mathsf{addr}^n$

$$\exists x \cdot \mathit{decide}(x) \rightarrow \mathit{decided} = \mathrm{F} \tag{19.1}$$

$$\mathit{decide}(x) \rightarrow \mathbf{X}(\mathit{decision} = x \wedge \mathit{decided} = \mathrm{T}) \tag{19.2}$$

$$\bigwedge_{c \in \Gamma_c} \exists \vec{v_c} \cdot c(\vec{v_c}) \wedge \mathit{decision} = x \wedge \mathit{decided} = y \rightarrow \mathbf{X}(\mathit{decision} = x \wedge \mathit{decided} = y) \tag{19.3}$$

$$\bigwedge_{c \in \Gamma_c} \exists \vec{v_c} \cdot c(\vec{v_c}) \wedge \mathit{initial} = x \wedge \mathit{known} = \vec{k} \rightarrow \mathbf{X}(\mathit{initial} = x \wedge \mathit{known} = \vec{k}) \tag{19.4}$$

$$\bigwedge_{b \in \Gamma_{l_b}} \exists \vec{k}, \vec{v_b} \cdot b(\vec{k}, \vec{v_b}) \rightarrow \mathit{decided} = \mathrm{F} \wedge \mathit{initial} = \mathit{decision} \wedge \mathit{known} = \vec{k} \tag{19.5}$$

**End**

Figure 4.2: Schematic specification of unreliable agreement processes.

occurrence of a failure. This means that we are dealing with crash failures.

The processes that attempt to reach distributed agreement in any system are considered to be unreliable in the precise sense specified above. We represent this fact through a morphism $\iota$ connecting the schematic specification of unreliable processes $\textsc{UnrelProc}$ to that of agreement processes $\textsc{UnrelAgmProc}$. Apart from the symbols dealing with the occurrence of failures, the language of agreement processes is also required to contain symbols to treat the occurrence of a decision, *decide*, the value initially proposed by the process, *initial*, and the possibly agreed boolean value, *decision*. As in the case of failures, we also adopt a boolean attribute *decided* to denote whether or not a decision action has already happened. Furthermore, a list of processes which is known to be attempting to reach agreement is kept as the value of the attribute *known*. The (schematic) axioms in Figure 4.2 are similar to those specifying the occurrence of failures. We need to stress at this point that many other presentations would also be suitable to deal with the consensus problem — what is important in this situation is the set of properties enjoyed by the specified objects — but we prefer the presentation above to facilitate our exposition.

Now we can formalise the dynamic nature of the problem. Upon proper initialisation, the four properties listed above are required to hold. The safe part of these properties may be formally stated as follows:

$$\mathit{Init}(\vec{k}, \vec{n})\mathbf{:} \quad \bigvee_{b \in \Gamma_{l_b}} \forall i \cdot \exists n, \vec{v_b} \cdot n.\mathbf{new}(b, k_i, \vec{k}, \vec{v_b}) \wedge \qquad \text{(Initialisation)}$$
$$\bigwedge_{c \in \Gamma_{e - e_b}} \mathbf{XG}(\forall l \cdot \exists i, \vec{v_c} \cdot l.\mathbf{send}\ c, k_i, \vec{v_c}\ (\rightarrow) \exists j \cdot l = n_j);$$

$Term(\vec{k})$**:** $\forall i \cdot k_i.failed = \mathrm{F} \rightarrow k_i.decided = \mathrm{T};$ (Termination)

$Agm(\vec{k})$**:** $\exists v \cdot \forall i \cdot k_i.failed = \mathrm{F} \rightarrow k_i.decision = v;$ (Agreement)

$Integ(\vec{k})$**:** $\forall i \cdot \exists v \cdot k_i.decide(v) \rightarrow \mathbf{XG}(\nexists v \cdot k_i.decide(v));$ (Integrity)

$Val(\vec{k})$**:** $\forall i, v \cdot k_i.decision = v \rightarrow \exists j \cdot k_j.initial = v.$ (Validity)

Using the formulas above and the logical properties of actors, the consensus problem can be formulated as the validity of the following liveness property, for each $\vec{k}$ such that $\mathsf{len}\ \vec{k} \geq 2$ and each $\vec{n}$ :

$$Init(\vec{k}, \vec{n}) \rightarrow \mathbf{F}(Term(\vec{k}) \wedge Agm(\vec{k}) \wedge Integ(\vec{k}) \wedge Val(\vec{k})) \qquad (4.1.1)$$

Note that this sentence may be obtained as a result of reasoning according to the rely-guarantee discipline described in the previous chapter.

The solution of the problem above clearly depends on the particular set of properties specified as part of each presentation. Fischer *et al.* (1985) gives a semantic proof that there is no solution if the mode of interaction is purely asynchronous. However, we cannot guarantee that this is the case using only the previous schematic presentations, since even completely synchronous communication can be specified in terms of asynchronous message passing, as illustrated in the preceding chapter. On the other hand, a totally synchronous solution based on our schematic presentations obtained via an application of our synchrony transformation is also impossible as a target process failure would imply in a source process deadlock in any communication. Many partial synchrony solutions, which depend on fine grain decisions concerning the adopted mode of interaction, are studied by Dolev *et al.* (1987). Despite these deterministic solutions, if all the processes in the system may fail and there is no synchrony involved, the problem is impossible. In these circumstances, a solution based solely on the actor model cannot be proposed.

One elegant way of hiding the fine grain decisions concerning the mode of interaction between agreement processes is the assumption of failure detectors as proposed by Chandra and Toueg (1996). Each process is assumed to have access to a local failure detection object. Such objects keep a list of processes that are suspected to have failed, which is dynamically updated by inclusion or removal. Failure detectors can make mistakes but are required to obey some *completeness* and *accuracy* properties demanding, for instance, that eventually every process that crashes is always suspected by some reliable process and that some reliable process is eventually never suspected by any of the processes that do not crash. A number of failure detectors can be proposed obeying these properties.

Note that failure detection objects are not assumed to crash. This distinction between objects that may and may not crash, together with the assumption that failures are to be detected amongst the set of given processes, establishes a separation of the involved system components into base-level and meta-level objects. Note that failure detectors are about the system itself. In this way, they are not to have any connection with the problem domain, naturally belonging to a meta-level of the system. Because some abstraction similar to failure detectors is required in order to hide the underlying mode of interaction and this can be captured in terms of meta-level architectures, it seems that it is reasonable to consider the latter notion as necessary in the design of extensible systems. In the next section, we propose a novel way of designing meta-level architectures.

## 4.2 The Design of Meta-Level Architectures

The central point in designing meta-level architectures is to draw an explicit boundary between base-level and meta-level functionality. This separation of what concerns the problem domain and the system itself is normally accomplished stating a set of non-interference properties. Saeki *et al.* (1993), for instance, requires that base and meta-level objects do not communicate explicitly. Venkatasubramanian and Talcott (1993) require that meta-level objects communicate with each other via message passing but manipulate base objects as data structures. This kind of organisation is illustrated in Figure 4.3.

Each object in the base-level is associated to some meta-level object wherein base-level state and events are represented. This gives the meta-level access to the features of the base-level, which in our case includes the hidden part of the state related to message buffering. The hidden state of the respective objects is represented as the dark parts of Figure 4.3. Meta and base level are also required to be aware of the mail addresses of each other and this enables their interaction.

We formalise the previous intuition about the representation of base-level information in the meta-level through the following definition:

**Definition 4.2.1 (Base-level representation)** Given actor specifications $\Phi_i = (\Delta_i,\ \Psi_i)$, $i \leq i \leq 2$, $\Phi_2$ is said to *represent a meta-level* of $\Phi_1$ if there is a specification morphism $\Phi_1 \xrightarrow{\tau} \Phi_2$ such that:

1. $\tau$ maps attribute and actions of $\Delta_1$ into $\Delta_2$, i.e., associates to pairs of distinct symbols in each of these families of $\Delta_1$ pairs of distinct $\Delta_2$ symbols;

2. there are flexible terms $meta_\tau \in Term(\Delta_1)_{\mathsf{addr}}$ and $base_\tau \in Term(\Delta_2)_{\mathsf{addr}}$ such that $\tau(meta_\tau) \neq base_\tau$;

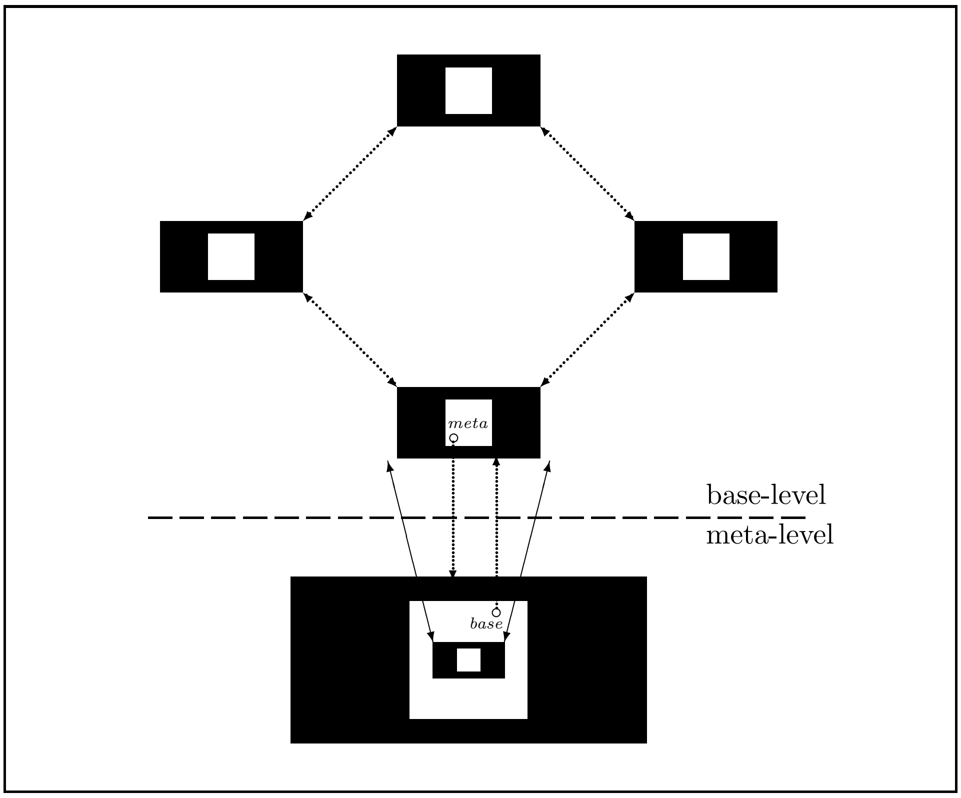


Figure 4.3: Relationship between base and meta-level objects.

3. $\Phi_2 \vdash \tau(p)$ iff $\Phi_1 \vdash p$ for each $p \in \Phi_1 \cup Ax_{\Phi_1}$.

Given the morphism $\tau$, we say that $\tau$ *represents* the base-level $\Phi_1$ in $\Phi_2$. $\square$

The requirement that representation morphisms be injective on attribute and action symbols in (1) captures the intuition that the architecture meta-level keeps a full representation of all the behavioural characteristics of the base-level. In order to allow unlimited access to base-level information in the meta-level, as previously described, we also have to lift the restriction that some symbols in the base-level representation are hidden. Condition (2) is to guarantee that base and meta-level objects can be made aware of the mail address of each other and the representation process does not preclude these objects from being distinct, having different mail addresses. The last condition (3) says that the properties of each object are the same when observed from either level of the architecture.

Base-level representation just ensures that it is possible for base and meta levels to co-exist in the same system. To guarantee that this certainly happens, we also need to make some assumptions on the way objects fulfilling each of these roles are related:

**Definition 4.2.2 (Meta-relation)** Given actors denoted by $\{x, y\} \subset \mathcal{V}_{\mathsf{addr}}$ and specifications $\Phi_i = (\Delta_i,\ \Phi_i)$, $1 \leq i \leq 2$, such that $\Phi_2$ represents the meta-level of $\Phi_1$, $y$ in the $\Phi_2$-community is said to be a *meta-level object* of the *base-level object* $x$ in the $\Phi_1$-community if for some $\tau$ representing $\Phi_1$ in $\Phi_2$:

1. $x \neq y$;
2. $\bigwedge_{f \in \mathcal{A}-\mathcal{A}_i} \forall k, \vec{v_f} \cdot x.f(\vec{v_f}) = k \leftrightarrow y.\tau(f)(\vec{v_f}) = k$ and $\bigwedge_{c \in \Gamma - \Gamma_{l_b \cup in_b}} \forall \vec{v_c} \cdot x.c(\vec{v_c}) \leftrightarrow y.\tau(c)(\vec{v_c})$;
3. $x.\tau(meta_\tau).base_\tau = x$ and $y.base_\tau.\tau(meta_\tau) = y$. $\square$

The first condition prevents that the same base-level object be related to itself as part of the meta-level. The second one says that meta-level objects simulate the behaviour of their base-level counterparts. This is quite a strong requirement. For instance, it implies that identical messages are always dispatched to base and meta-level objects. There are ways of making this requirement more reasonable by increasing the amount of sharing allowed by the formalism — it may be possible to consider that the same message, with the same identification, is dispatched to both base and meta level — but we prefer to leave this treatment unspecified in order to deal with the problem in an abstract manner. Due to our mutual exclusion assumptions, the previous requirement also implies that meta-level objects will present some independent behaviour only when their respective base-level is inactive. Finally, the third condition says that base and meta-level objects know the mail addresses of each other. From the definitions above, we see that, to design meta-level architectures, we have to split the design in two parts as usual: a pair of specifications related by meta-representation is proposed and two existing objects are assumed to be always meta-related.

Note that, given a representation morphism, the assumption that two objects are meta-related can be stated using a finite conjunction of formulas. That the two objects exist and belong to distinct communities is simply ensured by relating the occurrence of their birth actions. The conditions (1-3) above are all stated in terms of single formulas. Therefore, it is feasible to write such a finitary conjunction as part of a rely-guarantee assertion.

Also note that our definitions are quite permissive concerning multiple connections between base and meta levels. For instance, it is possible that the same base-level object be directly related to several distinct meta-level objects. What is necessary to determine this situation, apart from the respective assumptions, is a set of representation morphisms, each requiring the existence of a distinct *meta* attribute. On the other hand, it is possible for the same meta-level object to represent several base-level objects. This happens because the attribute symbol *base* is not required to be in the image of any representation morphism. In this way, the same meta-level specification can determine many such attributes which are related to the respective base levels through their representation morphisms and the corresponding assumptions. Furthermore, by chaining these static and dynamic relationships, a finite hierarchy of meta-levels can be specified as part of the same system.

Let us return to the consensus problem. We sketch in what follows a solution based on the rigorous discipline proposed above. Our solution requires that the meta-level of each object, that is the respective failure detector, knows the meta-level address of all the other processes attempting to reach agreement. This is achieved by requiring that each of the $n$ agreement processes broadcasts the mail address of its meta-level object just after the occurrence of the respective birth action. Note that some of these messages may never be received since some failures may happen first. We also require that the interaction between a process and its local failure detector be synchronous and make the assumption that meta and base-level objects communicate only among themselves.

Each failure detector operates in asynchronous rounds, whose steps are determined by the consumption of self addressed messages. Each failure detector automatically knows if its process failed or not, due to our construction giving base-level access to these meta-level objects, and is always eventually enabled for delivery of any message pertaining only to the meta-level. At the end of each round, the failure detector updates its list of suspects with the information possibly received by other detectors and broadcasts to all the other failure detectors the list of processes known to have crashed. In this way, the information provided by each of these objects is always correct and accurate.

The agreement processes themselves also operate based on asynchronous rounds and keep lists of values that are known to be initially proposed by each process. In the beginning, this list contains only the value proposed by the local process. After $n-1$ rounds, wherein proposed values are broadcast and propositions from all processes that are not suspected are awaited, each process that has not crashed will be aware of a list of initially proposed values. During each round, the list of suspected processes is dynamically updated to reflect information provided by the local failure detector. In a second stage, the list kept by each process is broadcast and every received list is used together with the local list to compute a least common denominator that will replace the latter list. Again, the list of suspects is updated while new lists of values are expected. After this stage, there will be agreement on the list of proposed values among the processes that have not crashed. Finally, each reliable process decides the first value of this list. The system clearly reaches consensus after this procedure.

The solution above is essentially that proposed by Chandra and Toueg (1996), particularised with our specification of failure detectors derived from the assumed meta-level architecture. Therein, a proof can be found that property (4.1.1) is valid considering these assumptions.

## 4.3 Computational Reflection

The assumption of meta-level architectures is extremely powerful. Even without making reference to the hidden symbols in the representation of base-level objects, to solve the consensus problem in the previous section we could design perfect failure detectors, in the sense that they are always correct, accurate and never make mistakes. Therefore, it appears to be natural to ask ourselves if the assumption of reflective architectures would be even more desirable.

This question may be given two answers. Considering that we are interested in extensible systems, reflective architectures are certainly desirable since they permit behavioural changes of system features at any architecture level. For instance, in a reflective text editor like Emacs (Stallman 1981), it is possible to extend the system providing not only a new way of cutting and pasting text based on menus, apart from keystroke commands, but also new ways of making this extension, through forms or changing a configuration file, to mention a few possibilities, and this chain of extensions could be infinitely iterated.

On the other hand, if we consider our interest in systematic software development, it does not make sense to assume reflective architectures. It would be impossible to prove any non-trivial safety property concerning the previously mentioned reflective text editor, for instance that the editing session is not terminated unless the text is saved first, because this and other properties would depend on the extensions performed at run time. Such extensions would have to be reflected in the specification of the system itself. It would also be impossible to define a non-trivial satisfaction relation between such specifications and programs because the programming language would have to be reflective as well. Therefore, the assumption of reflection is inappropriate in systematic software development. This has already been identified by Agha *et al.* (1993), for instance. We need to clarify, however, that designing a reflective architecture is different from assuming the existence of one such an architecture in the design of a distinct system: the design of a reflective architecture does not need to assume the existence of one such an architecture. In the realm of programming language design, this has already been shown by Wand and Friedman (1988).

Part of the argument above can also be explained in terms of our formal definitions in the previous section. Reflective architectures are those where the chain of meta-levels is not finite. In this way, to give formal treatment to the assumption of such an architecture, we would have to provide an infinite number of morphisms connecting each pair of architecture levels and write an infinitary conjunction of assumptions of meta-related objects. This is clearly impossible using the finitary first-order logical systems studied in this thesis, but may not represent a problem if an infinitary logic based on $\mathbf{L}_{\omega_1\omega}$, say, is adopted.

## 4.4 Summary and Related Work

In this chapter, we have argued that the assumption of meta-level architectures should be considered necessary in the design of extensible systems. Through an example, we showed that meta-level architectures provide a direct and elegant solution to a specific problem which otherwise demands fine grain decisions concerning the mode of interaction between objects and is in some cases impossible. We proposed a design discipline that treats this assumption. We also argued that the assumption of reflective architectures is not compatible with the notion of systematic software development.

Maes (1987) was the first to propose a systematic study of computational reflection and to develop a reflective object-based programming language. Agha (1997) recognised the need of more expressive formalisms to deal with meta-level architectures. Venkatasubramanian and Talcott (1993) established a clear distinction between meta-level and reflective architectures, which we have followed, and developed semantic methods for reasoning about such architectures. The latter work is based on the actor model, as is our case.

Saeki *et al.* (1993) extended the specification language LOTOS with reflective facilities. In particular, the restriction that base-level representation must focus just on behavioural aspects of the system was reflected here in Definition 4.2.1 through the requirement that attribute and action symbols be fully represented. Simhi *et al.* (1996) proposed the use of state transition diagrams to enhance the design of reflective objects. Both works contrast with our view that reflection is not compatible with systematic software development and software design in particular.

It is also interesting to mention that all the work on distributed consensus has been developed in a semantic way (Fischer *et al.* 1985, Dolev *et al.* 1987, Chandra and Toueg 1996). Our formalism and discipline appear to provide a suitable proof-theoretic framework for dealing with this and related problems.

# Chapter 5

# Case Study: Location Management for Mobility

We are currently facing a radical change in the way users interact with software systems and in the underlying distributed software architectures. Thanks to the advent of technologies like cellular phones, personal digital assistants and active badges, users are no longer required to go to specific access points to take advantage of some locally provided functionality. Such devices have become increasingly more personal and can be carried by their owners. In turn, the respective software systems may now be used at any time and place, and can provide location dependent functionality such as ubiquitous message delivery, transportable user sessions and others (Harter and Hopper 1994). Software components which implement these features are identified by end users as extending the functionality provided at their current location. What is essentially novel in this completely new kind of operational environment is the very presence of *mobility*. The way to support the new requirements related to mobility is to manage location information.

The need to manage location information and mobility brings with it new problems to be addressed in the design of distributed systems. The autonomy and heterogeneity presented by mobile objects make it not only difficult but virtually impossible to account for many interesting features required in real implementations as part of any design. Moreover, to ensure that these systems are open, characteristics that depend on the current technology need to be abstracted away. In this context, specifications have to be supported by a formalism which is expressively rich enough to represent the remaining properties. VDM (Jones 1990) and Z (Spivey 1989), for instance, do not address at all the inherent concurrency of mobile systems. In some other cases, concurrency is actually treated but the development process is organised in terms of notions like processes (Milner 1983) or programs (Chandy and Misra 1988, Wilcox and Roman 1996), which certainly provide important insights on how an implementation should work but poorly support understanding and representing the problem domain in an organised manner. The use of object-based notions like attributes, actions and encapsulation as studied in the previous chapters seems to bridge this gap, but even then expressibility concerns arise since the basic notion of mobility has to be captured.

We have chosen as a case study in this chapter the design of a particular location management architecture for networks of mobile users and devices, as originally sketched in (Duarte 1997a), to illustrate in a more realistic situation the application of our formalism and design discipline. For simplicity, we ignore the important issues of dependability, authenticity and security (Spreitzer and Theimer 1994), concentrating just in the management of location information. We also abstract away many details that are essential to ensure reasonable performance (Lam *et al.* 1996). In the next section, we informally describe the requirements of location management applications. We then devote two sections to their design, namely their specification and verification. We conclude this chapter providing a comparison with related work.

## 5.1 Location Management: Requirements

A central problem in designing and implementing software systems for networks of mobile users and devices is how to manage distributed object locations. An extensive description of the problem can be found in the literature (cf. Harter and Hopper 1994, Leonhardt and Magee 1996, Spreitzer and Theimer 1994). In this section, we provide an informal list of requirements strictly imposed by mobility. In the next section, we discuss some design decisions based on this list and propose a formal specification for the corresponding mobile architecture.

We can classify the requirements for managing distributed object locations into three families, the first concerning the nature of location information and located objects, the second about the process of acquiring location information and the third on how to deal with it. In what follows, we provide a partial list of functional requirements:

1. *Location information* must be *dynamic*, in the sense that, at each time, it may be a distinct instance of a class[1] of objects;

2. *Location information* must be *mutable*, in the sense that, at each time, it may be an instance of a distinct class of objects;

[1]Here we consider the word *class* in a loose sense, just as a set of objects.

3. *Located objects* may be *users* or *devices*, at least;
4. *Location information acquisition* must be *unintrusive*, which means that the acquisition process cannot intrude user behaviour nor require user intervention;
5. *Location information acquisition* must offer support to *multiple location observations*, which means that simultaneous observations producing distinct location information for the same object may occur;
6. *Location information management* must support *indeterminacy*, which means that location information for some objects may not be available at some instant;
7. *Location information management* must offer support to *object naming*, which is the assignment of meaningless unique names to located objects.

The first two items should not be confused. While mobile object locations clearly may need to change as time passes, meaning that they are dynamic, it is not so obvious that they should also be mutable. This is because a location service may provide information with distinct accuracies or multiple services may be used, as observed by Leonhardt and Magee (1996). The requirement of unique object naming may be controversial, but appears to be the minimal condition to support properties not treated here such as authenticity and security.

## 5.2 Location Management in a Formal Setting

Before introducing definitions directly related to location management, we present in Figure 5.1 the specification of region tree nodes, particular instances of the spatial hierarchical data structures proposed by Samet (1984). These will be used in our design later on. At the top of the specification, we can see sort symbols denoting not only standard actor data types but also the four compass points (direc). Constant and operation symbols appear in the same statement. Each quadratic planar region is represented by a terminal node (*node*), which is created undivided ($bot = \mathrm{T}$), or by a root node (*root*). Terminal nodes may receive requests to divide themselves in sub-regions ( *split*) organised in $reg$ according to the directions of the compass points. Nodes need to be aware of their own mail address ($me$) and the name of a parent node ($pr$), when it exists. Eventually, continuations may be created ($ct$) to expect the answer of queries of region inclusion ($in$). Defined in this way, these are quaternary trees wherein nodes may be dynamically associated to more refined partitions of the plane.

**Actor** REGIONTREENODE

**data types** $\mathsf{addr}, \mathsf{bool}, \mathsf{int}, \mathsf{direc}$ $(\mathrm{T}, \mathrm{F} : \mathsf{bool}; 0, 1, 3 : \mathsf{int}; + : \mathsf{int} \times \mathsf{int} \rightarrow \mathsf{int}; \mathrm{N}, \mathrm{S}, \mathrm{E}, \mathrm{W} : \mathsf{direc})$

**attributes** $me, pr : \mathsf{addr}; reg : \mathsf{direc} \rightarrow \mathsf{addr}; up, bot : \mathsf{bool}; an : \mathsf{int}$

**actions** $ct(\mathsf{addr}^2)$ : **local birth**;
$root(\mathsf{addr}^5), node(\mathsf{addr}^2)$ : **local + extrn birth**;
$go, inc, updt(\mathsf{addr}^4)$ : **local computation**;
$ack(\mathsf{addr}^4)$ : **extrn message**;
$split(\mathsf{addr}), in(\mathsf{addr}^2), rpl(\mathsf{addr}^2, \mathsf{bool})$ : **local + extrn message**

**axioms** $n : \mathsf{addr}^4; k, p, q, r, s, t, u, x, y, z : \mathsf{addr}; d : \mathsf{direc}, v : \mathsf{int}, b : \mathsf{bool}$

$root(k, \vec{n}) \rightarrow me = k \wedge up = \mathrm{F} \wedge bot = \mathrm{F} \wedge reg = \vec{n} \wedge an = 0$ (20.1)

$node(k, p) \vee ct(k, p) \rightarrow me = k \wedge pr = p \wedge up = \mathrm{F} \wedge bot = \mathrm{T} \wedge reg[d] = k \wedge an = 0$ (20.2)

$root(k, \vec{n}) \vee node(k, p) \vee ct(k, p) \rightarrow \mathbf{X}(go)$ (20.3)

$go \wedge reg = \vec{n} \wedge an = v \rightarrow \mathbf{X}(reg = \vec{n} \wedge an = v \wedge up = \mathrm{T})$ (20.4)

$updt(\vec{n}) \wedge up = b \wedge an = v \rightarrow \mathbf{X}(up = b \wedge an = v \wedge bot = \mathrm{F} \wedge reg = \vec{n})$ (20.5)

$inc \wedge up = b \wedge reg = \vec{q} \wedge an = v \rightarrow \mathbf{X}(up = b \wedge reg = \vec{q} \wedge an = v + 1)$ (20.6)

$(go \vee updt(\vec{n}) \vee inc) \wedge me = p \wedge pr = q \rightarrow \mathbf{X}(me = p \wedge pr = q)$ (20.7)

$split(k) \wedge me = p \rightarrow \mathbf{X}(\exists \vec{q} \cdot updt(\vec{q}) \wedge \mathbf{new}(node, q_i, q_i, p) \wedge \mathbf{send}\ ack, k, \vec{q}\ ())$ (20.8)

$(in(k, p) \wedge r = k \vee rpl(r, s, \mathrm{T}) \wedge pr = p) \wedge me = k \rightarrow \mathbf{X}(\mathbf{send}\ rpl, p, r, k, \mathrm{T}\ ())$ (20.9)

$in(k, p) \wedge me = q \neq k \wedge bot = \mathrm{F} \wedge reg = \vec{r} \rightarrow \mathbf{X}(\exists!\, s \cdot \mathbf{new}(ct, s, q, p) \wedge \mathbf{send}\ in, r_i, k, s\ ())$ (20.10)

$in(k, p) \wedge me = q \neq k \wedge bot = \mathrm{T} \rightarrow \mathbf{X}(\mathbf{send}\ rpl, p, k, q, \mathrm{F}\ ())$ (20.11)

$rpl(k, p, \mathrm{F}) \wedge pr = r \wedge me = q \rightarrow \mathbf{X}(an = 3 \wedge \mathbf{send}\ rpl, r, k, q, \mathrm{F}\ (\vee) an \neq 3 \wedge inc)$ (20.12)

$\exists \vec{n}, \vec{p} \cdot \mathbf{new}(node, n_i, p_i, q) \vee updt(\vec{n}) \vee \mathbf{send}\ ack, r, \vec{n}\ (\leftarrow) split(r) \wedge me = q$ (20.13)

$inc \leftarrow \exists k, p \cdot rpl(k, p, \mathrm{F}) \wedge an \neq 3$ (20.14)

$\mathbf{send}\ rpl, k, p, q, \mathrm{T}\ (\leftarrow)(in(q, k) \wedge p = q \vee \exists s \cdot rpl(p, s, \mathrm{T}) \wedge pr = k) \wedge me = q$ (20.15)

$\mathbf{send}\ rpl, k, p, q, \mathrm{F}\ () \leftarrow (in(p, k) \wedge p \neq q \wedge bot = \mathrm{T} \vee \exists s \cdot rpl(p, s, \mathrm{F}) \wedge pr = k \wedge an = 3) \wedge me = q$
(20.16)

$\exists k \cdot \mathbf{new}(ct, k, q, r) \vee \mathbf{send}\ in, p_i, s, k\ (\leftarrow) in(s, r) \wedge me = q \neq s \wedge bot = \mathrm{F} \wedge reg = \vec{p}$ (20.17)

$up = \mathrm{T} \rightarrow \mathbf{FE}(\mathbf{deliv}\ (split, p)) \wedge \mathbf{FE}(\mathbf{deliv}\ (in, q, r)) \wedge \mathbf{FE}(\mathbf{deliv}\ (rpl, s, t, b))$ (20.18)

$up = \mathrm{T} \rightarrow \mathbf{FE}(split(t)) \wedge \mathbf{FE}(in(u, x)) \wedge \mathbf{FE}(rpl(y, z, b))$ (20.19)

**End**

Figure 5.1: Specification of region trees.

Based on the requirements list above, we make our first design decision following Harter and Hopper (1994) by using references to objects denoting geographic regions instead of dealing with location information directly. In this way, each located object acquires a new attribute (*loc*), which is to contain the mail address of an actor representing a region in a location space. Using region trees as in Figure 5.1 for this purpose, we treat both the dynamic and mutable character of location information with this decision: as the value of an attribute, such information can always be changed; as a reference, it does not constrain the shape and size of location observations. We make, however, the simplifying assumption that geographic regions are divided into disjoint squares, due to the structure of such trees. In a real global location system, it may be more appropriate to adopt a location space divided according to a spherical coordinate system with origin in the earth centre. Unique object naming is treated similarly,

**Actor** Sensor
  **data types** $\mathsf{addr}, \mathsf{bool}, \mathsf{int}$ $(\text{T}, \text{F} : \mathsf{bool}; 0, 1, \text{MAX} : \mathsf{int}; + : \mathsf{int} \times \mathsf{int} \to \mathsf{int})$
  **attributes** $me, srv, obj, id, loc : \mathsf{addr}; up : \mathsf{bool}; time : \mathsf{int}$
  **actions** $sens(\mathsf{addr}^5)$ : **local** + **extrn birth**;
    $go, reloc(\mathsf{addr}), set(\mathsf{int}), obs$ : **local computation**;
    $tick$ : **local** + **extrn message**;
    $detect(\mathsf{addr}^2), unreach(\mathsf{addr}^2)$ : **extrn message**
  **axioms** $n, p, q, r : \mathsf{addr}, v : \mathsf{int}, b : \mathsf{bool}$
  $sens(n,p,q,r,s) \to me = n \wedge srv = p \wedge obj = q \wedge id = r \wedge loc = s \wedge time = 0 \wedge up = \text{F}$ (21.1)
  $sens(n,p,q,r,s) \to \mathbf{X}(go \wedge \textbf{send } tick, n\ ())$ (21.2)
  $go \wedge me = n \wedge loc = q \wedge time = v \to \mathbf{X}(me = n \wedge loc = q \wedge time = v \wedge up = \text{T})$ (21.3)
  $reloc(n) \wedge me = p \wedge time = v \wedge up = b \to \mathbf{X}(loc = n \wedge me = p \wedge time = v \wedge up = b)$ (21.4)
  $set(v) \wedge me = n \wedge loc = q \wedge up = b \to \mathbf{X}(time = v \wedge me = n \wedge loc = q \wedge up = b)$ (21.5)
  $(go \vee reloc(n) \vee set(v)) \wedge srv = p \wedge obj = q \wedge id = r \to \mathbf{X}(srv = p \wedge obj = q \wedge id = r)$ (21.6)
  $obs \wedge me = n \wedge srv = p \wedge obj = q \wedge id = r \to \mathbf{X}(me = n \wedge srv = p \wedge obj = q \wedge id = r)$ (21.7)
  $obs \wedge time = v \wedge loc = n \wedge up = b \to \mathbf{X}(time = v \wedge loc = n \wedge up = b)$ (21.8)
  $obs \wedge srv = n \wedge loc = p \wedge (obj = q \vee id = q) \to \mathbf{X}(set(0) \wedge \textbf{send } detect, n, q, p\ ())$ (21.9)
  $tick \wedge time = \text{MAX} \wedge src = n \wedge obj = p \wedge loc = q \to \mathbf{X}(set(0) \wedge \textbf{send } unreach, n, p, q\ ())$
(21.10)
  $tick \wedge time \neq \text{MAX} \wedge time = v \to \mathbf{X}(set(v+1))$ (21.11)
  $\textbf{send } detect, n, p, q\ (\leftarrow) obs \wedge srv = n \wedge loc = p \wedge (obj = q \vee id = q)$ (21.12)
  $\textbf{send } unreach, n, p, q\ (\leftarrow) tick \wedge time = \text{MAX} \wedge src = n \wedge obj = p \wedge loc = q$ (21.13)
  $set(v) \leftarrow v = 0 \wedge (obs \vee tick \wedge time = \text{MAX}) \vee v = time + 1 \wedge tick \wedge time \neq \text{MAX}$ (21.14)
  $up = \text{T} \to \mathbf{FE}(\textbf{deliv } (tick)) \wedge \mathbf{FE}(tick)$ (21.15)
**End**

Figure 5.2: Specification of sensors.

requiring the existence of a naming attribute (*id*) in each named object.

In order to treat the requirements related to location information acquisition and management, we first adopt the specification of sensors in Figure 5.2. Each sensor should be created with knowledge of a location service mail address (*srv*) and is responsible for producing sequential observations (*obs*) of a named located object (*obj*) in a specific region (*loc*). Sensors are mobile as well and detect themselves in the monitored region (21.9). We omit their straightforward generalisation to deal with the observation of several distinct objects.

Each sensor keeps an internal clock which evolves due to a stream of self-addressed *tick* messages initiated just after the actor is created. Upon creation, the resulting occurrence of a computation *go* makes the actor ready for the delivery and consumption of such messages. The clock is reset, $set(0)$, after MAX cycles or when the user is observed (21.9 and 21.10). Axiom (21.14) guarantees that resets do not happen in other occasions. Indeterminacy is treated by this clocking mechanism, which signs to the location service that the user is unreachable (*unreach*) whenever observations do not happen before the deadline MAX (21.10). A *detect* message with the user location is sent to the service

**Actor** MobileAgent
**data types** $\mathsf{addr}, \mathsf{bool}$ $(\text{T}, \text{F} : \mathsf{bool})$
**attributes** $me, id, loc, to : \mathsf{addr}; up, fwg, nul : \mathsf{bool}$
**actions** $ag(\mathsf{addr}^3)$ : **local + extrn birth**;
$redir(\mathsf{addr})$ : **local computation**;
$sub(\mathsf{addr}^2)$ : **extrn message**;
$fwd(\mathsf{addr}), mv(\mathsf{addr}^2), cp(\mathsf{addr}^3)$ : **local + extrn message**
**axioms** $n, p, q, r, s, t : \mathsf{addr}; b : \mathsf{bool}$

$ag(n,p,q) \rightarrow me = n \wedge id = p \wedge loc = q \wedge fwg = \text{F} \wedge to = n$ (22.1)

$ag(n,p,q) \rightarrow \mathbf{X}(go)$ (22.2)

$go \wedge fwg = b \wedge to = n \rightarrow \mathbf{X}(up = \text{T} \wedge fwg = b \wedge to = n)$ (22.3)

$redir(n) \wedge up = b \rightarrow \mathbf{X}(fwg = \text{T} \wedge to = n \wedge up = b)$ (22.4)

$(go \vee redir(n)) \wedge me = p \wedge id = q \wedge loc = r \rightarrow \mathbf{X}(me = p \wedge id = q \wedge loc = r)$ (22.5)

$fwd(n) \rightarrow \mathbf{X}(redir(n))$ (22.6)

$mv(n,p) \wedge fwg = \text{F} \wedge me = q \wedge id = r \rightarrow \mathbf{X}(redir(q) \wedge \textbf{send } cp, n, p, q, r\ ())$ (22.7)

$mv(n,p) \wedge fwg = \text{T} \wedge to = q \rightarrow \mathbf{X}(\textbf{send } mv, q, n, p\ ())$ (22.8)

$cp(n,p,q) \wedge loc = r \rightarrow \mathbf{X}(\exists! s \cdot \textbf{new}(ag, s, s, q, r) \wedge \textbf{send } fwd, p, s\ () \wedge \textbf{send } sub, n, s, r\ ())$
(22.9)

$redir(n) \leftarrow fwd(n) \vee \exists p, q \cdot (mv(p,q) \wedge me = n \wedge fwg = \text{F})$ (22.10)

$\exists n, p \cdot \textbf{new}(ag, n, p, q, r) \vee \textbf{send } fwd, s, n\ (\vee)\textbf{send } sub, t, n\ (\leftarrow) cp(t, s, q) \wedge loc = r$ (22.11)

$\textbf{send } mv, n, p, q\ (\leftarrow) mv(p,q) \wedge to = n \wedge fwg = \text{T}$ (22.12)

$\textbf{send } cp, n, p, q, r\ (\leftarrow) mv(n,p) \wedge me = q \wedge id = r \wedge fwg = \text{F}$ (22.13)

$up = \text{T} \rightarrow \mathbf{FE}(\textbf{deliv } (cp, n, p, q)) \wedge \mathbf{FE}(\textbf{deliv } (mv, r, s)) \wedge \mathbf{FE}(\textbf{deliv } (fwd, t))$ (22.14)

$up = \text{T} \rightarrow \mathbf{FE}(cp(n,p,q)) \wedge \mathbf{FE}(mv(r,s)) \wedge \mathbf{FE}(fwd(t))$ (22.15)

**End**

Figure 5.3: Simplified specification of mobile agents.

otherwise (21.9). Multiple location observations are obtained by many sensors concurrently dealing with the same located object and by the (fair) merge of observation messages delivered to the location service. Unintrusivity is also enforced as no causal connection between the production of observations and user behaviour is imposed.

If we realise the sensors of Figure 5.2 as optical devices connected to the architecture through radio frequency links, for instance, software mobility arises only when located object agents are considered. Such agents are meant to follow located objects through the architecture providing location dependent functionality such as ubiquitous message delivery and transportable user sessions (Spreitzer and Theimer 1994). Although we leave this additional functionality unspecified here, we present a specification of mobile agents in Figure 5.3.

We choose to capture the handover process of mobile objects as localised agent replication. A mobile agent $q$ may receive a request from $p$ to move to the location of another agent $n$ $(q.mv(n,p))$, presumably located closer to the object $q$ represents. If an agent is currently moving to a new location $(fwg = \text{T})$, such requests will be delayed by self-forwarding until the agent finishes to move (22.8).

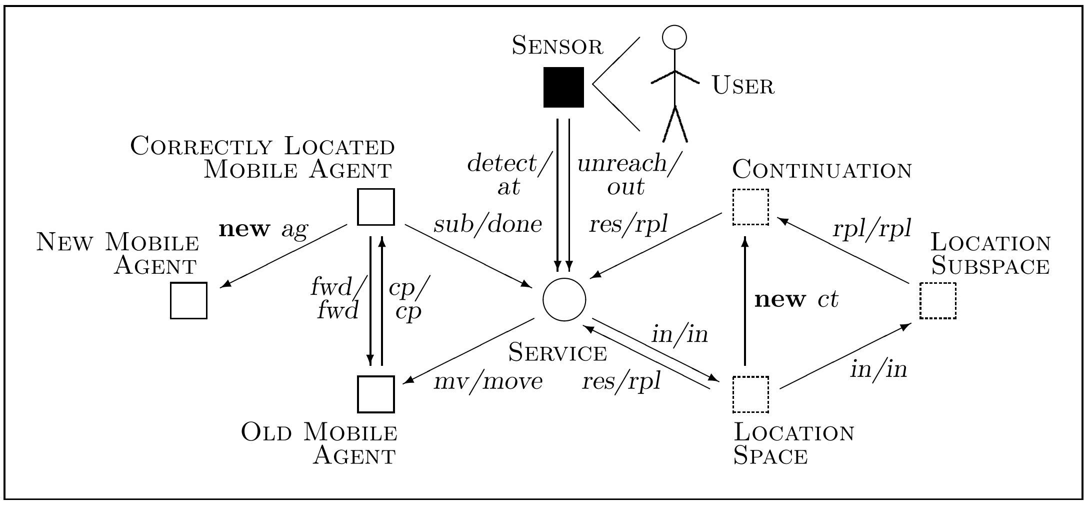


Figure 5.4: Internal event flow of the mobile architecture.

In order to move, the original agent $q$ issues a request for the correctly located agent $n$ to create a local copy (*cp*) of $q$ (22.7), supplying in the message any required information for the copy (here, in particular, just its logical name *id*). After consuming this kind of replication request, an agent creates the desired replica and notifies both the original agent and the requesting service that the located object representative can be substituted, through the messages *fwd* and *sub* (22.9), respectively.

To ensure coordination between sensors and agents, a location service must guarantee that the asynchronous messages they exchange are correctly addressed and ordered. This situation is explained by the diagram in Figure 5.4. Once a located object is detected in a region (*at*), the location service has to find among the registered objects a corresponding mobile agent in the region to request the creation of a replica of the moving agent therein. The location space is recurrently queried (*in*) until such an agent is found. Then, the service requests the agent of the relocated object to move to the place of the correctly located agent (*move*). At the end, the service is notified (*done*) so that the old agent can be discarded and new movement requests can be processed.

Since the location service has to associate located object names (*id*) to mobile agents, to keep track of their location (*loc*) and to put agents in contact to support mobility, we consider that server nodes providing compartmentalised bits of this functionality, one for each located object, are organised in circular lists, adopting the specification in Figure 5.5. Each server node also records if there is no location information available for the object ($st$ = NL). This

**Actor** SERVER

**data types** $\mathsf{addr}, \mathsf{bool}, \mathsf{status}$ ($\text{T}, \text{F} : \mathsf{bool}; \text{OK}, \text{NL}, \text{MV} : \mathsf{status}$)

**attributes** $me, id, loc, xt, ag : \mathsf{addr}; st : \mathsf{status}$

**actions** $srv(\mathsf{addr}^5)$ : **local** + **extrn birth**;

$ch(\mathsf{addr}^3, \mathsf{bool})$ : **local computation**;

$mrq(\mathsf{addr}^3), ack(\mathsf{addr}), ins(\mathsf{addr}^4), done(\mathsf{addr}^2)$ : **local** + **extrn message**;

$?(\mathsf{addr}^2), res(\mathsf{addr}^2, \mathsf{bool}), at(\mathsf{addr}^2), out(\mathsf{addr}^2)$ : **local** + **extrn message**;

$move(\mathsf{addr}), in(\mathsf{addr}^2), @(\mathsf{addr}^2)$ : **extrn message**

**axioms** $n, p, q, r, s, t, u, x : \mathsf{addr}, v : \mathsf{status}$

$$srv(n,p,q,r,s) \rightarrow me = n \wedge xt = p \wedge id = q \wedge loc = r \wedge ag = s \wedge st = \text{OK} \tag{23.1}$$

$$ch(n,p,q,v) \rightarrow \mathbf{X}(xt = n \wedge loc = p \wedge ag = q \wedge st = v) \tag{23.2}$$

$$ch(n,p,q,b) \wedge me = r \wedge id = s \rightarrow \mathbf{X}(me = r \wedge id = s) \tag{23.3}$$

$$ins(n,p,q,r) \wedge xt = s \wedge loc = t \wedge ag = u \wedge st = v \rightarrow \mathbf{X}(\exists x \cdot \mathbf{new}(srv,x,x,s,n,p,q) \wedge ch(x,t,u,v)) \tag{23.4}$$

$$ins(n,p,q,r) \wedge xt = s \rightarrow \mathbf{X}(\exists t \cdot \mathbf{new}(srv,t,t,s,n,p,q) \wedge \mathbf{send}\ ack,r,t\ ()) \tag{23.5}$$

$$mrq(n,p,q) \wedge xt = r \wedge loc = s \wedge ag = t \rightarrow \mathbf{X}(\exists!\, u \cdot \mathbf{new}(srv,u,q,r,p,n,t) \wedge \mathbf{send}\ in,s,p,u\ ()) \tag{23.6}$$

$$?(n,p) \wedge id = n \wedge me = q \wedge loc = r \rightarrow \mathbf{X}(st = \text{OK} \wedge \mathbf{send}\ @,p,n,r\ () \vee st \neq \text{OK} \wedge \mathbf{send}\ ?,q,n,p\ ()) \tag{23.7}$$

$$?(n,p) \wedge id \neq n \wedge xt = q \rightarrow \mathbf{X}(\mathbf{send}\ ?,q,n,p\ ()) \tag{23.8}$$

$$at(id,p) \wedge me = q \wedge xt = r \wedge ag = s \wedge (loc \neq p \wedge st = \text{OK} \vee st = \text{NL}) \rightarrow \mathbf{X}(\mathbf{send}\ mrq,r,s,p,q\ ()) \tag{23.9}$$

$$at(n,p) \wedge id = n \wedge loc = q \wedge ag = s \wedge (p \neq q \wedge st = \text{OK} \vee st = \text{NL}) \rightarrow \mathbf{X}(ch(r,q,s,\text{MV})) \tag{23.10}$$

$$at(n,p) \wedge xt = q \wedge (id \neq n \vee st = \text{MV}) \rightarrow \mathbf{X}(\mathbf{send}\ at,q,n,p\ ()) \tag{23.11}$$

$$out(n,p) \wedge id = n \wedge xt = q \wedge loc = r \wedge ag = s \rightarrow \mathbf{X}(ch(q,r,s,\text{NL})) \tag{23.12}$$

$$out(n,p) \wedge id \neq n \wedge xt = q \rightarrow \mathbf{X}(\mathbf{send}\ out,q,n,p\ ()) \tag{23.13}$$

$$res(n,p,\text{T}) \wedge loc = s \wedge me = q \wedge ag = t \rightarrow \mathbf{X}(\mathbf{send}\ move,s,t,q\ ()) \tag{23.14}$$

$$res(n,p,\text{F}) \wedge me = r \wedge xt = s \wedge loc = t \wedge id = u \rightarrow \mathbf{X}(\mathbf{send}\ mrq,s,t,u,r\ ()) \tag{23.15}$$

$$done(n,p) \wedge xt = q \rightarrow \mathbf{X}(ch(q,p,n,\text{F})) \tag{23.16}$$

⋮

*and usual axioms for readiness, absence of unsolicited responses and enabledness*

⋮

**End**

Figure 5.5: Specification of location service nodes.

knowledge is used to postpone until the object location becomes known (23.7 and 23.8) the answer to location queries, using the message symbols @ and *?*.

Every message addressed to the location service circulates around the linked list until the node with correct identity is found. In case an observation (*at*) from a sensor arrives carrying a new object location (23.9), a request for the rest of the list to find some agent placed therein is issued aiming to support the movement to that location (*mrq*). For each registered located object, the location space is queried in a two step process: a continuation actor to process the query answer will be created (23.8), and this new actor will either request the relocated object agent to move (23.14) or will forward the query to the next list element (23.15).

The informal description of the relationship between each pair of specifica-

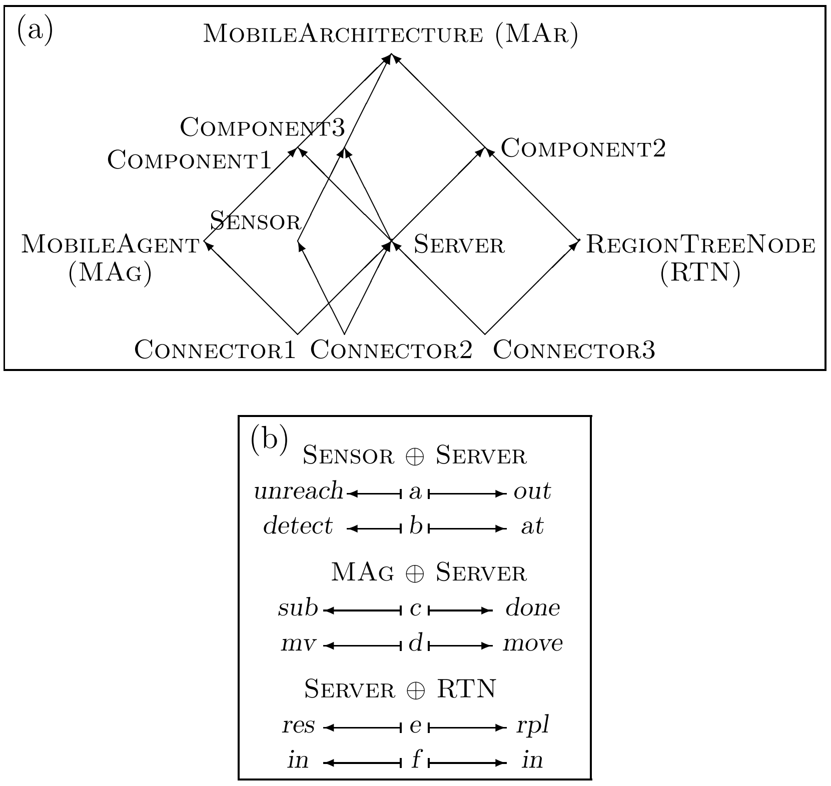


Figure 5.6: Composition of the architecture: Shared actors (a) and actions (b).

tions should not substitute their formal composition, which is still missing here. The diagram in Figure 5.4 gives a good clue on what remains to be defined: the “physical communication channels”, which are formally defined using specification morphisms. For each pair of specifications, individually represented by distinct geometric figures, that diagram shows how to relate their message symbols. For instance, the messages *mv* and *sub* of agents should be respectively associated with *move* and *done* of servers. Note that relating external to local symbols yields the direction of the message flow described above. Also observe in our example that we cannot produce a direct translation of the specification of agents into that of servers nor in the opposite direction. Therefore, to interconnect these entities we need to define mediating theory presentations to serve as connectors. Their nature is illustrated by the diagram in Figure 5.6.

To specify the linguistic structure of the mobile architecture in a formal manner, we call the mediating specifications in Figure 5.6.a Connectors. Each of them contains two external message symbols only (without axioms as well). We also provide translations including their contents after necessary renamings into the connected presentations. Taking connectors, connected specifications and the morphisms between them, the composite theory presentations are defined by pushout constructions. Defined in this way, each COMPONENT in the figure contains all the renamed symbols and the axioms of the connected specifications, but the symbols identified by the connectors are equalised. That is why a message *move* from servers can be understood as *mv* when it is delivered to an agent, for example, no matter its name in the composite component. The detailed definition of connectors and their morphisms appears in Figure 5.6.b.

## 5.3 Verifying Location Management Properties

The previous section introduced a set of composed specifications related to location management and described the intended behaviour of the specified objects when properly connected. In this section, we particularise our description providing more details about the dynamic configuration of our architecture. We make a number of simplifying assumptions to obtain a tractable example. In addition, we sketch the verification of some interesting properties. Rely-guarantee assertions are proposed below to capture these properties.

### 5.3.1 Location Space

We consider the existence of a non-trivial minimally divided location space represented by a tree of height one. This structure consists in a root and four child nodes, each of which denoting a quarter of the location plane associated to the respective compass point. The division of these location space regions is considered to be always forbidden. In addition, we assume that only child nodes may eventually communicate with continuation actors created by the root node to answer inclusion queries. Under these conditions, whenever a query is dispatched to a node in the location space, the query is answered eventually:

| **Assertion** *LOC* | |
|---|---|
| init | $k.\mathbf{new}(node, n_i, n_i, l)\ (i \in [1..4]), k.\mathbf{new}(root, l, l, \vec{n}),$ |
| | $\mathbf{G}(\forall p, q \cdot p = l \vee p \in \vec{n} \rightarrow \neg p.split(q))$ |
| rely | $\forall p, q \cdot (\exists r \cdot l.\mathbf{new}(cnt, q, l, r)) \rightarrow \mathbf{G}(\forall t \cdot (\exists r, s, v \cdot t.\mathbf{send}\ rpl, q, r, s, v\ ()) \rightarrow t \in \vec{n})$ |
| pre | $x.\mathbf{send}\ in, z, y, x\ (,)z = l \vee z \in \vec{n}$ |
| post | $\exists v \cdot z.\mathbf{send}\ rpl, y, x, v\ ()$ |

The derivability of the assertion above is justified by a case analysis argument. First note that the involved actors are always eventually enabled for delivery and consumption. Hence, if the recipient of a query is one of the child nodes, because the result of such queries depends solely on the state of the recipient itself (recall that nodes are assumed to remain undivided), the answer is locally produced eventually (20.9). If the recipient is the root node, the query is consumed, a continuation actor is created (20.10) and the query is dispatched to each child node. According to the preceding argument, a response is produced for each of these queries. In the end, due to our assumption, the continuation actor computes the query result after consuming only all child node responses (20.12).

### 5.3.2 Location Service

Now we can discuss the properties of the main components of our architecture. Here we analyse only a simple situation in which there are two locations actually populated and a pair of agents representing mobile objects. Another pair of fixed agents, one at each populated location, is also assumed to exit in order to support the handover process of mobile agents. All these objects are connected through a static circular network of server nodes where there is available a sensor per location and mobile agent.

To formalise the configuration above according to our previous informal descriptions, we also assume that there are actors which serve as mobile object identifiers. These are created through the birth action *name*, which is presumed to appear in a theory presentation connected to MAs. We use the logical uniqueness of their mail addresses to guarantee unique mobile object identification. The following definitions are also used in the assertions below:

$$\begin{aligned}
ploc(j) \text{ (nearest populated location)} &\overset{\text{def}}{=} (j - 1 \mathsf{\ div\ } 2) + 1 \\
nsv(j) \text{ (next server)} &\overset{\text{def}}{=} j + 1 \; (\mathsf{mod}\ 4) \\
psv(j) \text{ (previous server)} &\overset{\text{def}}{=} j - 1 \; (\mathsf{mod}\ 4) \\
fsv(j) \text{ (nearest server of fixed agent)} &\overset{\text{def}}{=} 2 * (j \mathsf{\ div\ } 3) + 1 \\
msv(j) \text{ (nearest server of mobile agent)} &\overset{\text{def}}{=} 2 * (j \mathsf{\ div\ } 3) + 2
\end{aligned}$$

We initially want to show that whenever a mobile object is observed, the respective observation message will eventually be consumed by the server node in charge of keeping track of the location of the agent:

**Assertion** $MAG$

init $\quad \mathsf{init}_{1..3}\text{-}LOC,$

$$j \in [1..4] \begin{cases} k.\mathbf{new}(\mathit{name}, na_j), k.\mathbf{new}(\mathit{name}, nss^{ploc(j)}_{fsv(j)}), \\ k.\mathbf{new}(\mathit{ag}, a_j, a_j, na_j, n_{ploc(j)}), k.\mathbf{new}(\mathit{srv}, sv_j, sv_j, sv_{nsv(j)}, na_j, n_{ploc(j)}, a_j), \\ k.\mathbf{new}(\mathit{sens}, ss^{ploc(j)}_{fsv(j)}, ss^{ploc(j)}_{fsv(j)}, sv_{fsv(j)}, nss^{ploc(j)}_{fsv(j)}, n_{ploc(j)}), \end{cases}$$

$\mathbf{G}(\nexists j \in [1..4], p, q, r, s \cdot sv_j.\mathit{ins}(p, q, r, s)),$

$\mathbf{G}(\nexists i, j \in [1..4], p \cdot a_{fsv(j)}.\mathit{mv}(p, q) \vee ss^{ploc(i)}_{fsv(j)}.\mathit{reloc}(p)),$

$\mathbf{G}(\forall j \in [1..4], p \cdot (\exists q, r \cdot p.\mathbf{send}\ \mathit{at}, sv_j, q, r\ ()) \rightarrow p = sv_{psv(j)} \vee \exists i \in [1..2] \cdot p = ss^{i}_{fsv(j)})$

rely $\quad \mathsf{rely}\text{-}LOC$

pre $\quad x.\mathit{obs}, \exists i, j \in [1..4] \cdot x = ss^{ploc(j)}_{fsv(i)}, x.\mathit{id} = y, x.\mathit{loc} = z$

post $\quad \exists g \in [1..4] \cdot sv_g.\mathit{at}(y, z) \wedge sv_g.\mathit{id} = y$

Since we know that each of the actors above is always eventually enabled for delivery and consumption, we can simply ignore this property again in the following justification of derivability of the assertion above. From our configuration assumption and (21.9), we infer that once an observation happens the respective message is dispatched to and eventually consumed by a server node. Two distinct situations may arise: the recipient node controls the relocated object agent, or, because this is not the case, the observation is sent to the next server node in the circular list (23.11). The observation message arrives at the appropriate node after reaching at most four such objects.

Now we wish to show that whenever a mobile object is observed, the respective agent eventually reaches the location of observation. Here we have to take into account two facts: only continuation actors used in querying the location space can decide whether to dispatch the query to the remainder of the list or to use the agent of the current server node, and the relocation process can only be completed due to a message received from the new object agent.

**Assertion** $MOV$

init $\quad \mathsf{init}\text{-}MAG,$

$\forall j \in [1..4] \cdot \mathbf{G}((\exists p \cdot sv_j.\mathit{at}(\mathit{id}, p) \wedge (sv_j.\mathit{loc} \neq p \wedge sv_j.\mathit{st} = \text{OK} \vee sv_j.\mathit{st} = \text{NL})) \rightarrow$

$(\nexists q, r \cdot sv_j.\mathit{done}(q, r))\mathbf{W}(\exists s, t, u, w \cdot sv_j.\mathit{ag}.\mathbf{new}(\mathit{ag}, s, t, u, w) \wedge \nexists q, r \cdot sv_j.\mathit{done}(q, r)))$

rely $\quad \mathsf{rely}\text{-}MAG,$

$\forall u \cdot (\exists j \in [1..4], p, q, r, s \cdot sv_j.\mathbf{new}(\mathit{srv}, u, sv_j, p, q, r, s)) \rightarrow$

$\quad \mathbf{G}(\forall t \cdot (\exists p, q, b \cdot t.\mathbf{send}\ \mathit{res}, u, p, q, b\ ()) \rightarrow \exists j \in [1..4] \cdot t = n_{ploc(j)})),$

$\forall r, s, t, u, j \in [1..4] \cdot (\exists p, q \cdot sv_j.\mathit{ag}.\mathbf{new}(\mathit{ag}, u, p, q, r)) \rightarrow$

$\quad (\neg sv_j.\mathit{done}(t, s))\mathbf{W}(t = r \wedge s = u))$

pre $\quad \mathsf{pre}\text{-}MAG$

post $\quad \exists g \in [1..4], s \cdot sv_{msv(g)}.\mathit{st} = \text{OK} \wedge sv_{msv(g)}.\mathit{ag} = s \wedge s.\mathit{id} = y \wedge s.\mathit{loc} = z$

We have to treat three different situations corresponding to the possible states of the recipient server node: the respective agent is ready to move ($st$ = OK), there is no location information available at the moment ($st$ = NL), or the agent is currently moving ($st$ = MV). We claim here and show below that whenever an agent is moving, this process is eventually completed and the respective server node is notified about this fact, returning to a ready state sometime in the future. So, it suffices to discuss the verification of the other two cases, which can both be treated as discussed in what follows.

If there is location information available saying that the agent is already at the right location, we reach our conclusion directly. Alternatively, we can use (23.9) and (23.6) to connect $LOC$ to $MAG$ and conclude that a movement request is dispatched to the remainder of the circular list (23.15). The movement request arrives at the appropriate server node after reaching at most four such objects. As a result, a movement request is dispatched to the relocated object agent (23.14). Because our assumptions prevent that a movement request arrives at an agent before a previous relocation process is completed, the request is consumed by the agent and a replication request is readily issued to the correctly located agent determined in the preceding querying process (22.6). Eventually, this message is consumed, the new agent for the relocated object is created and not only the old agent but also the respective server node are notified (22.8). In the end, the server enters into a ready state pointing to the properly located new agent (23.16).

Based on the previous assertion, we can also produce an interesting example using our general composition rule. Applying the substitution $[x\backslash x']$ throughout, we can generate another assertion analogous to the above. Composing these two assertions using our rule and requiring that $x \neq x'$, we can conclude that whenever two mobile object observations happen in parallel at different places, the respective agents will move to the involved locations eventually.

### 5.3.3 Other Properties of the Mobile Architecture

Location dependent functionality such as ubiquitous message delivery can be specified and verified based on the framework described above. Each mobile object agent should be able to query in terms of logical object identifiers ($id$) the location service for the location of the recipient. The message is dispatched to the fixed agent assumed to exist at that location. Upon receipt, such an agent either locally delivers the message to the recipient or forwards the message to another agent at the new location of the target object. In a similar context which does not consider temporary absence of location information, Sanders *et al.* (1997) outlines a proof that each message eventually reaches the recipient provided that this object eventually stops moving.

## 5.4 Summary and Related Work

As a means of illustrating that the formalism and discipline proposed in this thesis apply equally well to designing real-sized extensible systems, in this chapter we have shown how to approach object-based mobile systems. As it turns out, our logical system and the adopted rely-guarantee discipline can be directly applied without any modification or additional coding technique to capture mobility. Basically, our approach consists in annotating located objects with an additional attribute containing references to location objects, as suggested by Harter and Hopper (1994), and by assuming the existence of a network of fixed geographically distributed objects which can deliver localised replication of remote objects so as to support mobility. The advantage of approaching mobility in this extra-logical manner is that specification and verification can be carried out much in the way that we design any system using the same formalism.

A few related work can be gathered in the literature, most of which adopting the programming logic of UNITY (Chandy and Misra 1988). Sanders *et al.* (1997) concentrate in specifying and verifying the querying and routing algorithms of a mobile architecture. Their hierarchical organisation of the location space is similar to ours, but the problem is treated in a monolithic, unstructured manner, which we believe makes both specification and verification more difficult and error prone. Initial work developed by Wilcox and Roman (1996) on attempting to extend UNITY introduced mobility concepts just as part of the refinement process. If mobility arises in a set of requirements, that approach would not be so effective: initial specifications are required before any mobility aspect can be considered. Recently, the same research group has endowed UNITY with elaborated logical reasoning principles to tackle mobility (Roman *et al.* 1997, McCann and Roman 1998): each UNITY program is required to define a specific variable containing concrete locations and transient interactions between co-located objects may occur.

In the process calculi literature, mobility has also received a lot of attention, motivating the evolution of the static process configurations of SCCS (Milner 1983) to the dynamic ones of the $\pi$-calculus (Milner *et al.* 1992). We have provided evidence here that most of the features of $\pi$-calculus processes can also be specified using our logical system. For instance, we can simulate recursion creating continuation actors and exchanging asynchronous messages; dynamic data structures can be represented as objects and so on. More importantly, the requirements related to mobility receive a more refined treatment here as process in the $\pi$-calculus are modelled as terms while we adopt theory presentations to represent objects. On the one hand, it seems to be easier to define notions of simulation and reduction for processes, treating in this way the refinement and operational behaviour of the specified objects. On the other, here it is possible to specify and reason about mobile objects as first-class entities using our more expressive logical system, which we feel more appropriate to represent the real world. Orava and Parrow (1992) recognise that $\pi$-calculus specifications guarantee only that the specified features are possible, but these may not occur. This can only be avoided by adjoining modal or temporal connectives to process calculi. It would be interesting to compare our formalism to those proposed by Milner *et al.* (1993) in terms of expressive power.

# Chapter 6

# Concluding Remarks

In this thesis, we have characterised *extensible software systems* as those subject to functional or structural dynamic changes that may range over first class entities, which can be created, altered and referenced. We defined a first-order branching time logical system that seems to be expressively rich enough to specify and verify the properties of interest in this domain and particularised our system according to specific software development approaches that appear to enforce extensibility. In addition, we have argued in favour of a proof-theoretic way of dealing more effectively with the rigorous design of extensible systems. A number of contributions and ideas for further work are listed below as an outcome of our research.

## 6.1 Contributions

Extensible software systems have been increasingly demanded in practice, particularly as a means of bridging the gap between user requirements and actually provided functionality (see Bershad *et al.* (1995) for an example related to operating systems design). They have also become important in recent years with the advent of networking architectures that are inherently extensible. Although some recent work used the term extensibility to make reference to the capability of some software architectures of presenting extended functionality at run time, e.g. (Matsuoka 1993), we are not aware of any attempt at *characterising* this notion in full. We believe that it is important in software design to elucidate the meaning of this and related notions like openness, mobility and reconfigurability — the design space of extensible systems in the terminology of Wegner (1987) — for the sake of avoiding ambiguity and ensuring correctness.

Our characterisation appears to be theoretically important as it identifies logical features that are required in representing and verifying the notions mentioned above. At the programming level, some other authors have already been concerned with providing a formal account for extensibility:

> A programming language may be extended in data structures and/or in computation devices. Extension in data structures means the possibility of (run-time) modification of the data environment of a program. This extension can be of two kinds: static and dynamic. (Gergely and Úry 1991)

However, this distinction between data and computing units of extension seems to be appropriate only at lower abstraction levels. That is why we have preferred to develop instead a study of various software development approaches and on how they enforce extensibility. The way we represent extension, relying on hidden symbols, is similar to the aforementioned work:

> The extension in computation devices provides new control structures in our case. In order to introduce a new control structure we have to provide all necessary functions and relations required to realise the new control. Therefore, extension in computation devices can also be defined as a static extension of data structures. (Gergely and Úry 1991)

It is clear that the rigorous design of extensible systems requires a suitable logical system. In this thesis we have defined *a new first-order branching time logical system with equality* for this purpose. Our choice of a first-order system with equality stems from our desire to deal with communities of named objects which in general may reconfigure and grow in number without any *a priori* bound. A temporal logical system is chosen because we wish to represent many distinct modes of interaction between components, which may co-exist and thus behave concurrently. Finally, the assumption of branching flows of time appears to be an adequate way of talking about the existence of some behaviours in which a particular event occurs, typically in open system specifications, without committing all specified behaviours to present the same property. Many similar logics certainly exist, the temporal logic of actions developed by Lamport (1994) is the most notable example, but we prefer to adopt our own system for the reasons detailed in Chapter 2.

The raw logical system mentioned above was particularised in Chapter 3 according to two distinct software development approaches which enforce extensibility. We have proposed *an axiomatisation of the actor model*, which has been semantically studied by a number of authors. We also showed how to compose actor specifications using categorical constructions called co-limits and how to take advantage of the complex structure of actor descriptions to decompose the verification process. In particular, to verify global properties of actor components, we introduced a *new rely-guarantee discipline* based on simple temporal sentences. These developments constitute a suitable framework for open reconfigurable systems design, which indeed present all the aforementioned properties of extensible systems. In Chapter 3, another mode of interaction and a distinct software development approach were also shown to be representable in terms of actor specifications and morphisms, which shows that the actor model is expressively rich enough for many purposes in software design.

In Chapter 4, we sketched, by sticking to actor specifications but allowing references to their hidden symbols in a disciplined manner, *how meta-level objects can be given a formal treatment*. We argued that such a treatment should be regarded as necessary to ensure in an abstract and elegant way separation of concerns between base-level objects dealing with the problem domain and meta-level objects handling the system itself. On the other hand, we argued that one cannot rely on the assumption of completely general meta-level support, that of computational reflection, while defending systematic software development.

Finally, we showed in Chapter 5 that a family of *mobile systems can be rigorously designed* using the same formal constructions proposed in the rest of the thesis. We chose as a full example a location management architecture for mobile objects. This example served to illustrate that our formalism scales well to the treatment of real problems.

## 6.2 Further Work

Our logical system and its definition may give rise to interesting research. It appears to be worthwhile investigating if the axiom schemas listed in Figure 2.15 are independent from each other. As an outcome of this investigation, one should be able to assess if it is possible to reduce the number of schemas while retaining their intuitive meaning. Another direction for further work is to study if a slightly distinct semantics can be found so as to obtain a completeness result. This may be possible following the results already obtained by Andréka *et al.* (1995) concerning linear time logic. A more pragmatic continuation of this work is to provide automated support for software development by implementing our axiomatisation and a number of verification tactics using an interactive logical framework like Isabelle (Paulson 1994).

Another area that appears to deserve future investigation is the refinement of specifications in a way that ensures extensibility. We have studied approaches to the software process that enforce extensibility and can be captured at the specification level, but a long chain of refinement steps may be necessary before extensible software is obtained. In order to cater for the dynamic configuration of components specified using rely-guarantee constructions, their initialisation constraints and assumptions about their environment may be realised as coordination language constructs. In this case, specifications should be refined in the usual way to obtain a set of programs whenever possible. The challenge here is to define a systematic method which is also compositional in that implementations of any complex specification can be verified based on the verification that their components satisfy the constituents of the original specification.

Some other topics studied here which are not directly related to extensibility appear to have potential for practical application. It may be interesting to investigate how to capture multi-language proof calculi in terms of general logical structures as discussed in Chapter 2. Our synchrony transformation defined in Chapter 3 should be further investigated. In particular, to determine conditions ensuring that the transformation "preserves" deadlock freedom remains an open problem. In addition, it would be interesting to apply our discipline for designing meta-level architectures of Chapter 4 in other situations and assess if more general definitions can be proposed.

# Appendix I

# Useful Theorems

Formal proofs of the theorems stated in this appendix may be obtained directly from the author.

## I.1 Classical Propositional Logic

Postulating the axiomatization of *classical propositional logic* ($CPL$) discussed in Section 2.3, the following theorems over $\Delta \in \mathsf{obj}\ \mathbf{Sig}^{CPL}$ are provable:

**(HS)** $\{(p \to q), (q \to r)\} \vdash_{\Delta}^{CPL} p \to r$ (hypothetical syllogism)

**(REFL)** $\vdash_{\Delta}^{CPL} p \to p$ (reflexivity)

**(EXP)** $\vdash_{\Delta}^{CPL} p \to ((p \to q) \to q)$ (expansion)

**(PERM)** $\vdash_{\Delta}^{CPL} (p \to (q \to r)) \to (q \to (p \to r))$ (permutation)

**(LTRAN)** $\vdash_{\Delta}^{CPL} (p \to q) \to ((r \to p) \to (r \to q))$ (left transitivity)

**(RTRAN)** $\vdash_{\Delta}^{CPL} (p \to q) \to ((q \to r) \to (p \to r))$ (right transitivity)

**(CONT)** $\vdash_{\Delta}^{CPL} (p \to (p \to q)) \to (p \to q)$ (contraction)

**(NEG-L)** $\vdash_{\Delta}^{CPL} p \to (\neg p \to q)$

**(DOUB)** $\vdash_{\Delta}^{CPL} \neg\neg p \to p$ (double negation)

**(NEG-R)** $\vdash_{\Delta}^{CPL} (p \to q) \to ((p \to \neg q) \to \neg p)$

**(CONP)** $\vdash_{\Delta}^{CPL} (p \to q) \to (\neg q \to \neg p)$ (contrapositive)

**(OR-L)** $\{p \to q, r \to q\} \vdash_{\Delta}^{CPL} p \vee r \to q$

**(OR-R)** $\{p \to q\} \vdash_{\Delta}^{CPL} p \to q \vee r$ [or $\{p \to q\} \vdash_{\Delta}^{CPL} p \to r \vee q$]

**(AND-L)** $\{p \to q\} \vdash_{\Delta}^{CPL} p \wedge r \to q$ [or $\{p \to q\} \vdash_{\Delta}^{CPL} r \wedge p \to q$]

**(AND-R)** $\{p \to q, p \to r\} \vdash_{\Delta}^{CPL} p \to q \wedge r$

**(AND-E)** $\{p \wedge q\} \vdash_{\Delta}^{CPL} p$ [or $\{p \wedge q\} \vdash_{\Delta}^{CPL} q$]

**(AND-I)** $\{p, q\} \vdash_{\Delta}^{CPL} p \wedge q$

**(IFF-RL)** $\{p \to q, q \to p\} \vdash_{\Delta}^{CPL} p \leftrightarrow q$

**(IFF-E)** $\{p \leftrightarrow q\} \vdash_{\Delta}^{CPL} p \to q$ [or $\{p \leftrightarrow q\} \vdash_{\Delta}^{CPL} q \to p$]

**(DM)** $\vdash_{\Delta}^{CPL} \neg(p \vee q) \leftrightarrow \neg p \wedge \neg q$ [or $\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$] (De Morgan)

**(DIST-OA)** $\vdash_{\Delta}^{CPL} p \vee (q \wedge r) \leftrightarrow (p \vee q) \wedge (p \vee r)$[1]
**(DIST-AO)** [or $p \wedge (q \vee r) \leftrightarrow (p \wedge q) \vee (p \wedge r)$]
(distribution of $\vee$ over $\wedge$)

**(DIST-IFA)** $\vdash_{\Delta}^{CPL} (p \to (q \wedge r)) \leftrightarrow (p \to q) \wedge (p \to r)$
**(DIST-IFO)** [or $(p \to (q \vee r)) \leftrightarrow (p \to q) \vee (p \to r)$]
(distribution of implication over $\wedge$ and $\vee$)

**(REPL-CPL)** $\{x \leftrightarrow y\} \vdash_{\Delta}^{CPL} p[q\backslash x] \leftrightarrow p[q\backslash y]$ (replacement)

## I.2 Propositional Linear Time Logic

Postulating the axiomatization of *linear time propositional logic* ($PLTL$) discussed in Section 2.4, the following theorems over $\Delta \in \mathsf{obj}\ \mathbf{Sig}^{PLTL}$ are provable:

**(REPL-PLTL)** $\{x \leftrightarrow y\} \vdash_{\Delta}^{PLTL} p[q\backslash x] \leftrightarrow p[q\backslash y]$ (replacement)

**(DIST-ORV)** $\vdash_{\Delta}^{PLTL} p\mathbf{V}r \vee q\mathbf{V}r \leftrightarrow (p \vee q)\mathbf{V}r$ (distribution of $\mathbf{V}$ over $\vee$)

**(DIST-ANDV)** $\vdash_{\Delta}^{PLTL} p\mathbf{V}(q \wedge r) \leftrightarrow p\mathbf{V}q \wedge p\mathbf{V}r$ (distribution of $\mathbf{V}$ over $\wedge$)

**(IDEM-F)** $\vdash_{\Delta}^{PLTL} \mathbf{FF}p \leftrightarrow \mathbf{F}p$ (idempotence of $\mathbf{F}$)

**(IDEM-G)** $\vdash_{\Delta}^{PLTL} \mathbf{G}p \leftrightarrow \mathbf{GG}p$ (idempotence of $\mathbf{G}$)

**(DUAL-GF)** $\vdash_{\Delta}^{PLTL} \mathbf{F}(\neg p) \leftrightarrow \neg\mathbf{G}p$ (duality between $\mathbf{G}$ and $\mathbf{F}$)

**(REFL-G)** $\vdash_{\Delta}^{PLTL} \mathbf{G}p \to p$ (reflexivity of $\mathbf{G}$)

**(MON-G)** $\vdash_{\Delta}^{PLTL} \mathbf{G}(p \to q) \to (\mathbf{G}p \to \mathbf{G}q)$ (monotonicity of $\mathbf{G}$)

[1] A sentence with $p$ at the right-hand side of each sub-formula is also provable.

**(RPL-GX)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{G}p \rightarrow \mathbf{X}p$

**(EXP-GX)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{G}p \rightarrow \mathbf{XG}p$

**(G⊤)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{G}\top$

**(NEG-V⊤)** $\vdash_{\triangle}^{PLTL}$ $\neg(\bot\mathbf{V}\top)$

**(NEG-V⊥)** $\vdash_{\triangle}^{PLTL}$ $\neg(\bot\mathbf{V}\bot)$

**(FUN-X)** $\vdash_{\triangle}^{PLTL}$ $\neg\mathbf{X}p \leftrightarrow \mathbf{X}(\neg p)$ (functionality of **X**)

**(MON-X)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{X}(p \rightarrow q) \rightarrow (\mathbf{X}p \rightarrow \mathbf{X}q)$ (monotonicity of **X**)

**(MON-GX)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{G}(p \rightarrow q) \rightarrow (\mathbf{X}p \rightarrow \mathbf{X}q)$

**(DIST-ANDX)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{X}(p \wedge q) \leftrightarrow \mathbf{X}p \wedge \mathbf{X}q$ (distribution of **X** over $\wedge$)

**(FIX-V)** $\vdash_{\triangle}^{PLTL}$ $q\mathbf{V}p \leftrightarrow \mathbf{X}(q \vee p \wedge q\mathbf{V}p)$ (fixed point of **V**)

**(FIX-U)** $\vdash_{\triangle}^{PLTL}$ $p\mathbf{U}q \leftrightarrow q \vee (p \wedge \mathbf{X}(p\mathbf{U}q))$ (fixed point of **U**)

**(FIX-F)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{F}p \leftrightarrow p \vee \mathbf{XF}p$ (fixed point of **F**)

**(FIX-G)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{G}p \leftrightarrow p \wedge \mathbf{XG}p$ (fixed point of **G**)

**(COM-GX)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{GX}p \leftrightarrow \mathbf{XG}p$ (commutativity of **G** and **X**)

**(COM-FX)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{FX}p \leftrightarrow \mathbf{XF}p$ (commutativity of **F** and **X**)

**(RPL-UF)** $\vdash_{\triangle}^{PLTL}$ $p\mathbf{U}q \rightarrow \mathbf{F}q$

**(MON-GF)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{G}(p \rightarrow q) \rightarrow (\mathbf{F}p \rightarrow \mathbf{F}q)$

**(DIST-ORF)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{F}(p \vee q) \leftrightarrow \mathbf{F}p \vee \mathbf{F}q$ (distribution of **F** over $\vee$)

**(DIST-ANDG)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{G}(p \wedge q) \leftrightarrow \mathbf{G}p \wedge \mathbf{G}q$ (distribution of **G** over $\vee$)

**(DIST-ANDF)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{F}(p \wedge q) \rightarrow \mathbf{F}p \wedge \mathbf{F}q$ (distribution of **F** over $\wedge$)

**(DIST-ORG)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{G}p \vee \mathbf{G}q \rightarrow \mathbf{G}(p \vee q)$ (distribution of **G** over $\vee$)

**(LIN-FX)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{F}p \wedge \mathbf{F}q \rightarrow \mathbf{F}(p \wedge q) \vee \mathbf{F}(p \wedge \mathbf{XF}q) \vee \mathbf{F}(q \wedge \mathbf{XF}p)$

**(LIN-G)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{G}(\mathbf{G}p \rightarrow q) \vee \mathbf{G}(\mathbf{G}q \rightarrow p)$

**(DIST-ORGF)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{GF}(p \vee q) \leftrightarrow \mathbf{GF}p \vee \mathbf{GF}q$

**(DIST-ANDFG)** $\vdash_{\triangle}^{PLTL}$ $\mathbf{FG}(p \wedge q) \leftrightarrow \mathbf{FG}p \wedge \mathbf{FG}q$

**(COM-FG)** $\vdash_{\Delta}^{PLTL}$ $\mathbf{FG}p \rightarrow \mathbf{GF}p$

**(MON-GU)** $\vdash_{\Delta}^{PLTL}$ $\mathbf{G}(p \rightarrow q) \rightarrow (p\mathbf{U}r \rightarrow q\mathbf{U}r)$
[or $\mathbf{G}(p \rightarrow q) \rightarrow (r\mathbf{U}p \rightarrow r\mathbf{U}q)$]

**(MON-GW)** $\vdash_{\Delta}^{PLTL}$ $\mathbf{G}(p \rightarrow q) \rightarrow (p\mathbf{W}r \rightarrow q\mathbf{W}r)$
[or $\mathbf{G}(p \rightarrow q) \rightarrow (r\mathbf{W}p \rightarrow r\mathbf{W}q)$]

**(RPL-WUF)** $\vdash_{\Delta}^{PLTL}$ $p\mathbf{U}q \leftrightarrow p\mathbf{W}q \wedge \mathbf{F}q$

**(TRAN-W)** $\{p \rightarrow q\mathbf{W}r, r \rightarrow q\mathbf{W}s\} \vdash_{\Delta}^{PLTL}$ $p \rightarrow q\mathbf{W}s$ (transitivity of $\mathbf{W}$)

The set of axiom schemes {**DUAL-GF**, **REFL-G**, **MON-G**, **RPL-GX**, **EXP-GX**, **FUN-X**, **MON-X**, **A10-G**, **FIX-U**, **RPL-UF**} together with **R1-MP** corresponds precisely to the propositional part of the axiomatization of the temporal logic of programs proposed in (Manna and Pnueli 1983).

## I.3 Propositional Branching Time Logic

Postulating the axiomatization of *branching time propositional logic* ($PBTL$) discussed in Section 2.5, the theorems over $\Delta \in \mathsf{obj}\ \mathbf{Sig}^{PBTL}$ below are provable:

**(DUAL-AE)** $\vdash_{\Delta}^{PBTL}$ $\mathbf{E}(\neg p) \leftrightarrow \neg\mathbf{A}p$ (duality between $\mathbf{A}$ and $\mathbf{E}$)

**(REPL-PBTL)** $\{x \leftrightarrow y\} \vdash_{\Delta}^{PBTL}$ $p[q\backslash x] \leftrightarrow p[q\backslash y]$ (replacement)

**(E-R)** $\vdash_{\Delta}^{PBTL}$ $p \rightarrow \mathbf{E}p$

**(MOD-B)** $\vdash_{\Delta}^{PBTL}$ $p \rightarrow \mathbf{AE}p$

**(CANC-EA)** $\vdash_{\Delta}^{PBTL}$ $\mathbf{EA}p \rightarrow p$ (cancelation of $\mathbf{EA}$)

**(IDEM-A)** $\vdash_{\Delta}^{PBTL}$ $\mathbf{A}p \leftrightarrow \mathbf{AA}p$ (idempotence of $\mathbf{A}$)

**(MON-AE)** $\vdash_{\Delta}^{PBTL}$ $\mathbf{A}(p \rightarrow q) \rightarrow (\mathbf{E}p \rightarrow \mathbf{E}q)$

**(DIST-ORE)** $\vdash_{\Delta}^{PBTL}$ $\mathbf{E}(p \vee q) \leftrightarrow \mathbf{E}p \vee \mathbf{E}q$ (distribution of $\mathbf{E}$ over $\vee$)

**(DIST-ANDA)** $\vdash_{\Delta}^{PBTL}$ $\mathbf{A}(p \wedge q) \leftrightarrow \mathbf{A}p \wedge \mathbf{A}q$ (distribution of $\mathbf{A}$ over $\wedge$)

**(DIST-ANDE)** $\vdash_{\Delta}^{PBTL}$ $\mathbf{E}(p \wedge q) \rightarrow \mathbf{E}p \wedge \mathbf{E}q$ (distribution of $\mathbf{E}$ overr $\wedge$)

**(DIST-ORA)** $\vdash_{\Delta}^{PBTL}$ $\mathbf{A}p \vee \mathbf{A}q \rightarrow \mathbf{A}(p \vee q)$ (distribution of $\mathbf{A}$ over $\vee$)

**(COM-AG)** $\vdash_{\Delta}^{PBTL}$ $\mathbf{AG}p \rightarrow \mathbf{GA}p$ (commutativity of $\mathbf{G}$ and $\mathbf{A}$)

**(COM-XA)** $\vdash^{PBTL}_{\Delta}$ $\mathbf{AX}p \to \mathbf{XA}p$ (commutativity of **A** and **X**)

**(COM-EF)** $\vdash^{PBTL}_{\Delta}$ $\mathbf{FE}p \to \mathbf{EF}p$ (commutativity of **E** and **F**)

**(COM-EX)** $\vdash^{PBTL}_{\Delta}$ $\mathbf{XE}p \to \mathbf{EX}p$ (commutativity of **E** and **X**)

**(IND-AG)** $\vdash^{PBTL}_{\Delta}$ $\mathbf{AG}(p \to \mathbf{X}p) \to (p \to \mathbf{XAG}p)$ (branching induction)

## I.4 Classical First-Order Logic

Postulating the axiomatization of *classical first-order logic* ($FOL$) discussed in Section 2.6, the following theorems over $\Delta \in \mathsf{obj}\ \mathbf{Sig}^{BFOL}$ are provable:

**(ALL-E)** $\vdash^{FOL}_{\Delta}$ $\forall x \cdot p[x] \to p$

**(MON-∀)** $\vdash^{FOL}_{\Delta}$ $\forall x \cdot (p \to q) \to (\forall x \cdot p \to \forall x \cdot q)$
(monotonicity of $\forall$)

**(GEN-∀)** $\{p\} \vdash^{FOL}_{\Delta}$ $\forall x \cdot p$

**(DUAL-∀∃)** $\vdash^{PBTL}_{\Delta}$ $\forall x \cdot (\neg p) \leftrightarrow \neg \exists x \cdot p$ (duality between $\forall$ and $\exists$)

**(REPL-FOL)** $\{x \leftrightarrow y\} \vdash^{FOL}_{\Delta}$ $p[q\backslash x] \leftrightarrow p[q\backslash y]$ (replacement)

**(MON-∀∃)** $\vdash^{PBTL}_{\Delta}$ $\forall x \cdot (p \to q) \to (\exists x \cdot p \to \exists x \cdot q)$

**(EXC-∀∃)** $\vdash^{FOL}_{\Delta}$ $\forall x \cdot (p[x] \to q) \leftrightarrow (\exists x \cdot p[x] \to q)$ provided that $x \notin Free(q)$.
[or $\exists x \cdot (p[x] \to q) \leftrightarrow (\forall x \cdot p[x] \to q)$]

**(MOV-IF∀)** $\vdash^{FOL}_{\Delta}$ $\forall x \cdot (p \to q[x]) \leftrightarrow (p \to \forall x \cdot q[x])$ provided that $x \notin Free(p)$.
**(MOV-IF∃)** [or $\exists x \cdot (p \to q[x]) \leftrightarrow (p \to \exists x \cdot q[x])$]

**(MOV-AND∀)** $\vdash^{FOL}_{\Delta}$ $\forall x \cdot (p \wedge q[x]) \leftrightarrow (p \wedge \forall x \cdot q[x])$ provided that $x \notin Free(p)$.
**(MOV-AND∃)** [or $\exists x \cdot (p \wedge q[x]) \leftrightarrow (p \wedge \exists x \cdot q[x])$]

**(DIST-AND∀)** $\vdash^{FOL}_{\Delta}$ $\forall x \cdot (p \wedge q) \leftrightarrow \forall x \cdot p \wedge \forall x \cdot q$
(distribution of $\forall$ over $\wedge$)

**(DIST-OR∃)** $\vdash^{FOL}_{\Delta}$ $\exists x \cdot (p \vee q) \leftrightarrow \exists x \cdot p \vee \exists x \cdot q$
(distribution of $\exists$ over $\vee$)

## I.5 Many-Sorted Logic with Equality

Postulating the axiomatization of *many-sorted logic with equality* ($MSFOL$) discussed in Section 2.6.1, the following theorems over $\Delta \in \mathsf{obj}\ \mathbf{Sig}^{MSFOL}$ are provable:

**(REPL-FOL)** $\{x \leftrightarrow y\} \vdash_{\Delta}^{MSFOL} p[q\backslash x] \leftrightarrow p[q\backslash y]$ (replacement)

**(REFL-EQ)** $\vdash_{\Delta}^{MSFOL} x = y \rightarrow y = x$ (reflexivity of equality)

**(TRAN-EQ)** $\vdash_{\Delta}^{MSFOL} x = y \wedge y = z \rightarrow x = z$ (transitivity of equality)

## I.6 First-Order Temporal Logic

Postulating the axiomatization of *linear time many-sorted first-order logic with equality* ($LTMSL$) discussed in Section 2.7, the following theorems over $\Delta \in \mathsf{obj}\ \mathbf{Sig}^{LTMSL}$ are provable:

**(FUN)** $\vdash_{\Delta}^{LTMSL} f(x_1, \ldots, x_n) = x \wedge f(x_1, \ldots, x_n) = y \rightarrow x = y$
for any $f \in Funct(\Delta) \cup Attr(\Delta)$ with $arity(f) = n$.

**(BARC-G)** $\vdash_{\Delta}^{LTMSL} \forall x \cdot \mathbf{G}p \leftrightarrow \mathbf{G}(\forall x \cdot p)$ (Barcan for $\mathbf{G}$)

**(BARC-X)** $\vdash_{\Delta}^{LTMSL} \forall x \cdot \mathbf{X}p \leftrightarrow \mathbf{X}(\forall x \cdot p)$ (Barcan for $\mathbf{X}$)
[or $\exists x \cdot \mathbf{X}p \leftrightarrow \mathbf{X}(\exists x \cdot p)$]

**(BARC-F)** $\vdash_{\Delta}^{LTMSL} \mathbf{F}(\exists x \cdot p) \leftrightarrow \exists x \cdot \mathbf{F}p$ (Barcan for $\mathbf{F}$)

**(BARC-GF)** $\vdash_{\Delta}^{LTMSL} \mathbf{GF}(\exists x \cdot p) \leftrightarrow \exists x \cdot \mathbf{GF}p$ (Barcan for $\mathbf{GF}$)

**(BARC-FG)** $\vdash_{\Delta}^{LTMSL} \forall x \cdot \mathbf{FG}p \leftrightarrow \mathbf{FG}(\forall x \cdot p)$ (Barcan for $\mathbf{FG}$)

**(BARC-A)** $\vdash_{\Delta}^{LTMSL} \forall x \cdot \mathbf{A}p \leftrightarrow \mathbf{A}(\forall x \cdot p)$ (Barcan for $\mathbf{A}$)

**(BARC-E)** $\vdash_{\Delta}^{LTMSL} \mathbf{E}(\exists x \cdot p) \leftrightarrow \exists x \cdot \mathbf{E}p$ (Barcan for $\mathbf{E}$)

# Appendix II

# Remaining Cases in the Proof of Soundness

We develop here the remaining cases of the soundness proof for our many-sorted first-order branching time logic with equality $MSBTL$. The corresponding axioms appear in Figure 2.15.

**(A2-I)** Suppose that (i) $(\theta, N, L, w_i) \models p \rightarrow (q \rightarrow r)$ and (ii) it is not the case that $(\theta, N, L, w_i) \models (p \rightarrow q) \rightarrow (p \rightarrow r)$. From (i) and two applications of **S3**, it is clear that (iii) if $(\theta, N, L, w_i) \models p$ then $(\theta, N, L, w_i) \models q$ implies $(\theta, N, L, w_i) \models r$. From (ii) and **S3**, (iv) $(\theta, N, L, w_i) \models p$ implies $(\theta, N, L, w_i) \models q$, (v) $(\theta, N, L, w_i) \models p$ but (vi) $(\theta, N, L, w_i) \models r$ does not hold. Applying (v) in (iv) results in $(\theta, N, L, w_i) \models q$, which in turn can be used together with (v) in (iii) to show that $(\theta, N, L, w_i) \models r$, contradicting (vi) in this way. Therefore, **S3** and the negation of our assumption allow us to conclude that $(\theta, N, L, w_i) \models (p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$;

**(A3-I)** Suppose that (i) $(\theta, N, L, w_i) \models \neg q \rightarrow \neg p$ and (ii) it is not the case that $(\theta, N, L, w_i) \models p \rightarrow q$. From (i) and **S3**, it is clear that (iii) $(\theta, N, L, w_i) \models \neg q$ implies $(\theta, N, L, w_i) \models \neg p$. Using (ii) and again **S3**, we also infer that (iv) $(\theta, N, L, w_i) \models p$ but (v) $(\theta, N, L, w_i) \models q$ does not hold. **S2** and (v) allow us to say that $(\theta, N, L, w_i) \models \neg q$, but using this fact in conjunction with (iii) shows that (iv) is contradicted. Therefore, by applying **S3** to the negation of our assumption, we conclude that $(\theta, N, L, w_i) \models (\neg q \rightarrow \neg p) \rightarrow (p \rightarrow q)$;

**(A5-GV)** Suppose that (i) $(\theta, N, L, w_i) \models \mathbf{G}(p \rightarrow q)$ and (ii) $(\theta, N, L, w_i) \models r\mathbf{V}p \rightarrow r\mathbf{V}q$ does not hold. From (ii) and **S3**, we have $(\theta, N, L, w_i) \models r\mathbf{V}p$ but $(\theta, N, L, w_i) \models r\mathbf{V}q$ is not the case. According to **S7**, this means that (iii) there is an $s_j \in \mathsf{dom}\ L$ with $L(w_i) < L(w_j)$ such that $(\theta, N, L, w_j) \models r$

and $(\theta, N, L, w_k) \models p$ for any $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(w_k) < L(w_j)$, and (iv) for every $w_m \in \mathsf{dom}\ L$ with $L(w_i) < L(w_m)$, $(\theta, N, L, w_m) \models r$ and $(\theta, N, L, w_n) \models q$ for any $w_n \in \mathsf{dom}\ L$ where $L(w_i) < L(w_n) < L(w_m)$ are not both true. In addition, the definition of satisfaction of $\mathbf{G}p$, (i) and **S3** show that (v) $(\theta, N, L, w_j) \models p$ implies $(\theta, N, L, w_j) \models q$ for any $w_j \in \mathsf{dom}\ L$ such that $L(w_i) \leq L(w_j)$. Applying the second half of (iii) in (v), we infer that $(\theta, N, L, w_o) \models q$ for every $w_o \in \mathsf{dom}\ L$ such that $L(w_o) < L(w_j)$. For $w_m = w_j$, when we conjoin this partial result to (iv), we obtain a contradiction. We conclude, from the negation of our assumption and **S3**, that $(\theta, N, L, w_i) \models \mathbf{G}(p \to q) \to (r\mathbf{V}p \to r\mathbf{V}q)$;

**(A7-V)** Suppose that (i) $(\theta, N, L, w_i) \models (p \wedge q\mathbf{V}p)\mathbf{V}p$. From (i), the definition of satisfaction of $\wedge$ and **S7**, we can see that (ii) there is $w_j \in \mathsf{dom}\ L$ with $L(w_i) < L(w_j)$ such that $(\theta, N, L, w_j) \models q \wedge q\mathbf{V}p$ and $(\theta, N, L, w_k) \models p$ for any $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(w_k) < L(w_j)$. Hence, from the first half of (ii) and the definition of satisfaction of $\wedge$, it is clear that (iii) there is an $w_m \in \mathsf{dom}\ L$ with $L(w_j) < L(w_m)$ such that $(\theta, N, L, w_m) \models q$ and $(\theta, N, L, w_n) \models p$ for any $w_n \in \mathsf{dom}\ L$ where $L(w_j) < L(w_n) < L(w_m)$. Because, $L(w_i) < L(w_j)$, we can certainly say from the first half of (iii) and the second half of (ii) that there is $w_j \in \mathsf{dom}\ L$ with $L(w_i) < L(w_j)$ such that $(\theta, N, L, w_j) \models q$ and $(\theta, N, L, w_k) \models p$ for any $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(w_k) < L(w_j)$. We conclude using the definition of satisfaction of $\wedge$, **S7** and **S3** that $(\theta, N, L, w_i) \models (p \wedge q\mathbf{V}p)\mathbf{V}p \to q\mathbf{V}p$;

**(A9-V)** Suppose that (i) $(\theta, N, L, w_i) \models (p \vee q)\mathbf{V}r$ and (ii) $(\theta, N, L, w_i) \models p\mathbf{V}r \vee q\mathbf{V}r$ is not the case. From (i), **S7** and the definition of satisfaction of $\vee$, (iii) there is $w_j \in \mathsf{dom}\ L$ with $L(w_i) < L(w_j)$, $(\theta, N, L, w_j) \models p$ or $(\theta, N, L, w_j) \models q$ and for any $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(s_k) < L(w_j)$, $(\theta, N, L, w_k) \models r$. Moreover, from (ii), **S7** and the definition of satisfaction of $\vee$, we infer that (iv) for every $w_l \in \mathsf{dom}\ L$ such that $L(w_i) < L(w_l)$, $(\theta, N, L, w_l) \models p$ and $(\theta, N, L, w_l) \models q$ are neither true or there is $w_m \in \mathsf{dom}\ L$ such that $L(w_i) < L(w_m) < L(w_l)$ where it is not the case that $(\theta, N, L, w_m) \models r$. In particular, (iv) holds for $w_l = w_j$, which contradicts (iii). Therefore, by applying **S3** to the negation of our assumption, we conclude that $(\theta, N, L, w_i) \models (p \vee q)\mathbf{V}r \to p\mathbf{V}r \vee q\mathbf{V}r$;

**(A11-X)** The definition of $\top$ and **S3** easily entail that $(\theta, N, L, w_i) \models \top$. In particular for $w_k \in \mathsf{dom}\ L$ such that $L(w_k) = L(w_i) + 1$, which exists and is unique due to the isomorphism between $\mathsf{dom}\ L$ and $\mathsf{cod}\ L$, $(\theta, N, L, w_j) \models \top$. Therefore, applying the definition of satisfaction of $\mathbf{X}p$, $(\theta, N, L, w_i) \models$

$\mathbf{X}\top$;

**(A14-A)** If $(\theta, N, L, w_i) \models \mathbf{A}p$ then $(\theta, N, L, (L^{-1} \circ L)(w_i)) \models p$, from **S8**, $L \cong L$ and the fact that each $L$ is invertible. But $(L^{-1} \circ L)(w_i) = I(w_i) = w_i$. Therefore, using **S8** and **S3**, $(\theta, N, L, w_i) \models \mathbf{A}p \to p$;

**(A18-Ebeg)** Assume that (i) $(\theta, N, L, w_i) \models \mathbf{beg}$ is false and (ii) there exists $L_i$ which agrees with $L$ on the state propositions satisfied up to $i$ such that $(\theta, N, L_i, (L_i^{-1} \circ L)(w_i)) \models \mathbf{beg}$. From (ii) and **S6**, we infer that $L_i((L_i^{-1} \circ L)(w_i)) = 0$. But $L_i \circ L_i^{-1} = I$, hence $L(w_i) = 0$. Using **S6** again, we reach $(\theta, N, L, w_i) \models \mathbf{beg}$, which contradicts (i). Therefore, based on **S3**, **S8**, we conclude that $(\theta, N, L, w_i) \models \mathbf{E}(\mathbf{beg}) \to \mathbf{beg}$;

**(A19-$\forall$)** Assume that $(\theta, N, L, w_i) \models \forall x \cdot p(x)$. From **S4**, for every $v \in \mathsf{cod}\ N$ and every assignment $N_v$ such that $N_v(y) = N(y)$ if $y \neq x$ or $N_v(y) = v$ otherwise, $(\theta, N_v, L, w_i) \models p$. This holds in particular for $v = [\![t]\!]^{\theta, N_v}(w_i)$ such that $t \in Term(\Delta)_s$ such that $Class(\Delta)(x) = s$. Therefore, using **S3**, by a structural induction argument on the notion of interpretation based on the definition of substitution and assignment, we conclude that $(\theta, N, L, w_i) \models \forall x \cdot p \to p[x \backslash t]$;

**(A21-EQ)** For any $t \in Term(\Delta)$, $[\![t]\!]^{\theta, N}(w_i) = [\![t]\!]^{\theta, N}(w_i)$, because terms have a functional interpretation. From **S5**, we conclude that $(\theta, N, L, w_i) \models (t = t)$;

**(A25-NEQG)** Assume that $(\theta, N, L, w_j) \models (t_1 \neq t_2)$ for $t_1$, $t_2$ free from any attribute symbol. In particular, for any $w_i \in \mathsf{dom}\ L$ such that $L(w_j) < L(w_i)$, $(\theta, N, L, w_i) \models (t_1 \neq t_2)$, due to **S2** and because $[\![t_1]\!]^{\theta, N}(w_i) \neq [\![t_1]\!]^{\theta, N}(w_j)$ and similarly for $t_2$. From the definition of satisfaction of $\mathbf{G}$, $\neq$ and **S3**, we conclude that $(\theta, N, L, w_i) \models (t_1 \neq t_2) \to \mathbf{G}(t_1 \neq t_2)$;

**(A27-EQA)** Assume that $(\theta, N, L, w_i) \models (t_1 = t_2)$ for $t_1$, $t_2$ free from attribute symbols. In particular, for any $L_i$ which agrees with $L$ on the state propositions satisfied up to $i$, $(\theta, N, L_i, (L_i^{-1} \circ L)(w_i)) \models (t_1 = t_2)$ due to **S2** and because $[\![t_1]\!]^{\theta, N}(w_i) = [\![t_1]\!]^{\theta, N}((L_i^{-1} \circ L)(w_i))$. Therefore, applying **S8** and **S3**, we conclude that $(\theta, N, L, w_i) \models (t_1 = t_2) \to \mathbf{A}(t_1 = t_2)$.

# Bibliography

Martín Abadi and Leslie Lamport (1994). Open systems in TLA. In: *Proc. 13th ACM Symposium on Principles of Distributed Computing (PODC'94)*, ACM Press 81–90.

Martín Abadi and Leslie Lamport (1995). Conjoining specifications. *ACM Transactions on Programming Languages and Systems* **17** (3) 507–534.

Martín Abadi and K. Rustan M. Leino (1997). A logic of object-oriented programs. In: Michel Bidoit (ed.), Proc. International Conference on the Theory and Practice of Software Development (TAPSOFT'97), *Springer-Verlag Lecture Notes in Computer Science* **1214** 682–696.

Martín Abadi and Stephan Merz (1996). On TLA as a logic. In: Manfred Broy (ed.), Deductive Program Design, *Springer-Verlag NATO ASI Series* 235–271.

Martín Abadi, Leslie Lamport, and Pierre Wolper (1989). Realizable and unrealizable specifications of reactive systems. In: Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca (eds.), Proc. 16th International Conference on Automata, Languages and Programming (ICALP'89), *Springer-Verlag Lecture Notes in Computer Science* **372** 1–17.

Martín Abadi (1989). The power of temporal proofs. *Theoretical Computer Science* **65** 35–83.

Samson Abramsky (1990). The lazy $\lambda$-calculus. In: David A. Turner (ed.), *Research Topics in Functional Programming*, Addison Wesley 65–117.

Gul Agha, Svend Frolund, Rajendra Panwar, and Daniel Sturman (1993). A linguistic framework for dynamic composition of dependability protocols. In: C. E. Landwehg, B. Randell, and L. Simoncini (eds.), Dependable Computing for Critical Applications 3, *Dependable Computing and Fault Tolerant Systems* **3** 345–363.

Gul Agha, Wooyoung Kim, and Rajendra Panwar (1994). Actor languages for specification of parallel computations. In: Guy E. Blelloch, K. Mani

Chandy, and Sridhar Jaganathan (eds.), Specification of Parallel Algorithms, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **18**, *American Mathematical Society* 239–258.

Gul Agha, Ian Mason, Scott Smith, and Carolyn Talcott (1997). A foundation for actor computation. *Journal of Functional Programming* **7** (1) 1–72.

Gul Agha (1986). *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press.

Gul Agha (1990). Concurrent object-oriented programming. *Communications of the ACM* **33** (9) 125–141.

Gul Agha (1997). Abstracting interaction patterns: A programming paradigm for open distributed systems. In: Elie Najm and Jean-Bertran Stefani (eds.), *Proc. 1st IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'96)*, Chapman and Hall 135–153.

Paulo S. C. Alencar, Don D. Cowan, and Carlos J. P. Lucena (1995). Abstract Data Views as a Formal Approach to Adaptable Software. In: *OOPSLA'95 Workshop on Adaptable and Adaptive Software.*

Bowen Alpern and Fred B. Schneider (1985). Defining liveness. *Information Processing Letters* **21** (4) 181–185.

Pierre America and Frank de Boer (1996). Reasoning about dynamically evolving process structures. *Formal Aspects of Computing* **6** (3) 269–316.

Hajnal Andréka *et al.* (1995). Effective temporal logics of programs. In: Leonard Bolc and Andrzej Szałas (eds.), *Time and Logic: A Computational Approach*, UCL Press 51–129.

Giuseppe Attardi and Maria Simi (1991). Reflections about reflection. In: James Allen, Richard Fikes, and Erik Sandewall (eds.), *Proc. 2nd. International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, Morgan Kaufmann 22–31.

Arnon Avron (1991). Simple consequence relations. *Information and Computation* **92** 105–139.

Michael Barr and Charles Wells (1990). *Category theory for computing science.* Prentice Hall.

Nuno Barreiro, José Fiadeiro, and Tom Maibaum (1995). Politeness in object societies. In: Roel Wieringa and Remco Feenstra (eds.), *Information Systems: Correctness and Reusability*, World Scientific 119–134.

Howard Barringer (1987). The use of temporal logic in the compositional specification of concurrent systems. In: Antony Galton (ed.), *Temporal Logics and their applications*, Academic Press 53–90.

David Basin and Seán Matthews (1996). Adding meta-theoretic facilities to first-order theories. *Journal of Logic and Computation* **6** (6) 835–849.

Brian Bershad *et al.* (1995). Extensibility, safety and performance in the SPIN operating system. In: *Proc. 15th ACM Symposium on Operating System Priciples (SOSP'95)*, ACM Press 267–284.

Antonio Cau and Pierre Collete (1996). Parallel composition of assumption-commitment specifications: A unifying approach for shared variable and distributed message passing concurrency. *Acta Informatica* **33** 153–176.

Maura Cerioli and José Meseguer (1997). May I borrow your logic? (Transporting logical structures along maps). *Theoretical Computer Science* **173** 311–347.

Tushar Deepak Chandra and Sam Toueg (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM* **43** (2) 225–267.

K. Mani Chandy and Jayadev Misra (1981). Proofs of networks of processes. *IEEE Transactions on Software Engineering* **7** (4) 417–426.

K. Mani Chandy and Jayadev Misra (1988). *Parallel Program Design, A Foundation.* Addison-Wesley.

C. C. Chang and H. J. Keisler (1977). *Model Theory.* North Holland, 2nd edition.

Bernardette Charron-Bost, Friedemann Mattern, and Gerard Tel (1996). Synchronous, asynchronous and causally ordered communication. *Distributed Computing* **9** 173–191.

Brian F. Chellas (1980). *Modal Logic: An Introduction.* Cambridge University Press.

Paolo Ciancarini and Chris Hankin (eds.) (1996). *Proc. 1st International Conference COORDINATION'96*, *Lecture Notes in Computer Science* **1061**. Springer-Verlag.

Manuel G. Clavel and José Meseguer (1996). Axiomatising reflective logics and languages. In: Gregor Kiczales (ed.), *Proc. Reflection'96.*

William D. Clinger (1981). *Foundations of Actor Semantics.* PhD thesis, Massachusetts Institute of Technology.

Pierre Collete (1994). Composition of assumption-commitment specifications in a UNITY style. *Science of Computer Programming* **23** 107–125.

John Darlington and Yike K. Guo (1995). Formalising actors in linear logic. In: Dilip Patel, Yuan Sun, and Shushma Patel (eds.), *Proc. International Conference on Object-Oriented Information Systems (OOIS'94)*, Springer-Verlag 37–53.

Ruy de Queiroz (1990). *Proof Theory and Computer Programming: The Logical Foundations of Computation.* PhD thesis, Department of Computing, Imperial College, London, UK.

Răzvan Diaconnescu, Joseph Goguen, and Petros Stefaneas (1993). Logical support for modularisation. In: Gérard Huet and Gordon Plotkin (eds.), *Logical Environments*, Cambridge University Press 83–130.

Danny Dolev, Cynthia Dwork, and Larry Stockmeyer (1987). On the minimal synchronism needed for distributed consensus. *Journal of the ACM* **34** (1) 77–97.

Carlos H. C. Duarte (1997). A proof-theoretic approach to the design of object-based mobility. In: Howard Bowman and John Derrick (eds.), *Proc. 2nd IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'97)*, Chapman and Hall 37–53. Canterbury, UK.

Carlos H. C. Duarte (1997). Towards a proof-theoretic foundation for actor specification and verification. In: Pierre-Yves Schobbens and Amedeo Cesta (eds.), *Proc. 4th Workshop on Formal Models of Agents (ModelAge'97)*, 115–128. Certosa di Pontignano, Italy.

Hans-Dieter Ehrich, Amilcar Sernadas, and Cristina Sernadas (1988). Objects, object types and object identity. In: Hartmut Ehrig (ed.), Categorical Methods in Computer Science with Aspects from Topology, *Springer Verlag Lecture Notes in Computer Science* **334** 142–156.

Hans-Dieter Ehrich (1982). On the theory of specification, implementation and parameterisation of abstract data types. *Journal of the ACM* **29** (1) 206–227.

Hartmut Ehrig and Bernd Mahr (1985). *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics, EATCS Monographs in Theoretical Computer Science* **6**. Springer Verlag.

E. Allen Emerson (1983). Alternative semantics for temporal logics. *Theoretical Computer Science* **26** 121–130.

E. Allen Emerson (1990). Temporal and modal logic. In: Jan van Leeuwen (ed.), Formal Models and Semantics, *Elsevier Handbook of Theoretical Computer Science* 996–1072.

José Fiadeiro and Tom Maibaum (1990). Towards object calculi. Technical report, Department of Computing, Imperial College, London.

José Fiadeiro and Tom Maibaum (1992). Temporal theories as modularisation units for concurrent systems specification. *Formal Aspects of Computing* **4** (3) 239–272.

José Fiadeiro and Tom Maibaum (1993). Generalising interpretations between theories in the context of ($\pi$-)institutions. In: Geoffrey Burn, Simon Gay, and Mark Ryan (eds.), Theory and Formal Methods 1993: Proceedings of the First Imperial College, Department of Computing Workshop, *Springer-Verlag* 126–147.

José Fiadeiro and Tom Maibaum (1994). Design structures for object-based systems. In: Sthepen Goldsack and Stuart Kent (eds.), *Formal Aspects of Object-Oriented Systems*, Prentice Hall.

José Fiadeiro and Tom Maibaum (1996). Interconnecting formalisms: Supporting modularity, reuse and incrementability. In: Gail E. Kaiser (ed.), *Proc. 3rd Symposium on Foundations of Software Engineering*, ACM Press 72–80.

José Fiadeiro and Tom Maibaum (1997). Categorical semantics of parallel program design. *Science of Computer Programming* **28** (2–3) 111–138.

José Fiadeiro and Amilcar Sernadas (1988). Structuring theories on consequence. In: Donald Sannella and Andrzej Tarlecki (eds.), Recent Trends in Data Type Specification, *Springer-Verlag Lecture Notes in Computer Science* **332** 44–72.

José Fiadeiro and Amilcar Sernadas (1990). Logics of modal terms for systems specification. *Journal of Logic and Computation* **1** (2) 187–227.

José Fiadeiro, Cristina Sernadas, Tom Maibaum, and Gunter Saake (1991). Proof-theoretic semantics of object-oriented specification constructs. In: Robert A. Meersman, Willian Kent, and Samit Khosla (eds.), *Proc. IFIP WG 2.6 Working Conference on Object-Oriented Databases: Analysis, Design and Construction*, North Holland 243–284.

José Fiadeiro (1996). On the emergence of properties in component-based systems. In: Martin Wirsing and Maurice Nivat (eds.), Proc. 5th International Conference on Algebraic Methodology and Software Technology

(AMAST'96), *Springer-Verlag Lecture Notes in Computer Science* **1101** 421–443.

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson (1985). Impossibility of distributed consensus with one faulty processor. *Journal of the ACM* **32** (2) 374–382.

Melvin Fitting (1983). *Proof Methods for Modal and Intuitionistic Logics*, *Studies in Epistemology, Logic and Methodology of Science* **169**. Reidel Dordrecht.

Robert W. Floyd (1967). Assigning meanings to programs. In: Jacob T. Schwartz (ed.), Mathematical Aspects of Computer Science, *American Mathematical Society Proc. Symposia in Applied Mathematics* **19** 19–32.

Harvey Friedman and Michael Sheard (1995). Elementary descent recursion and proof theory. *Annals of Pure and Applied Logic* **71** 1–45.

Dov Gabbay and Ruy de Queiroz (1992). Extending the Curry-Howard interpretation to linear, relevant and other resource logics. *Journal of Symbolic Logic* **5** (4) 1319–1365.

Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi (1980). On the temporal analysis of fairness. In: *Proc. 7th ACM Symposium on Principles of Programming Languages*, ACM Press 163–173.

Dov Gabbay, Ian Hodkinson, and Mark Reynolds (1994). *Volume I. Temporal Logic: Mathematical Foundations and Computational Aspects.* Oxford Science Publications.

Dov Gabbay (1981). An irreflexivity lemma with applications to axiomatisations of conditions on tense frames. In: Uwe Mönnich (ed.), Aspects of Philosophical Logic, *Synthese Library* **147**, *Kluwer Academic Publishers* 67–89.

David Garlan (1995). Research directions in software architecture. *ACM Computing Surveys* **27** (2) 257–261.

Gerhard Gentzen (1943). Beweisbarkeit und unbeweisbarkeit von anadagsfällen der transfiniten induktion in der reinen zahlentheorie. *Mathematische Annalen* **119** 140–161. English translation in (Szabo 1969).

Tamás Gergely and László Úry (1991). *First-Order Programming Theories*, *EATCS Monographs on Theoretical Computer Science* **24**. Springer-Verlag.

Jean-Yves Girard (1987). Linear logic. *Theoretical Computer Science* **50** (1) 1–102.

Fausto Giunchiglia and Luciano Serafini (1994). Multilanguage hierarchical logics (or: How can we do without modal logics). *Artificial Intelligence* **65** 1–42.

Kurt Gödel (1931). Übet formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatschefte für Mathematik und Physik* **38** 173–198. English translation in van Heijenoort (1967; 592–617).

Joseph A. Goguen and Rod M. Burstall (1992). Institutions: Abstract model theory for specification and programming. *Journal of the ACM* **39** (1) 95–146.

Joseph Goguen and José Meseguer (1981). Completeness of many-sorted equational logic. *Sigplan Notices* **16** (7) 24–37.

Robert Goldblatt (1979). *TOPOI: The categorial analysis of logic.* North Holland.

Robert Goldblatt (1992). *Logics of Time and Computation.* CSLI Publications, 2nd. edition.

Vassos Hadzilacos and Sam Toueg (1994). A modular approach to the specification and implementation of fault-tolerant broadcasts. Technical report 94-1425, Department of Computer Science, Cornell University.

Robert Harper, Donald Sannella, and Andrzej Tarlecki (1994). Structured presentations and logic representations. *Annals of Pure and Applied Logic* **67** 113–160.

Andy Harter and Andy Hopper (1994). A distributed location system for the active office. *IEEE Network* **8** (1) 62–70.

Carl Hewitt and Henry Baker (1977). Laws for communicating parallel processes. In: Gilchrist Bruce (ed.), *Information Processing 77: 7th IFIP World Congress*, Elsevier 987–992.

David Hilbert and Wilhelm Ackermann (1928). *Grundzüge Der Theoritischen Logik.* Springer. English translation by Robert E. Luck in "Principles of Mathematical Logic", Chelsea Publishing Company, 1950.

Charles A. R. Hoare (1978). Communicating sequential programs. *Communications of the ACM* **21** (8) 667–677.

Charles A. R. Hoare (1985). *Communicating Sequential Processes.* Prentice-Hall.

Kohei Honda and Mario Tokoro (1991). An object calculus for asynchronous communication. In: Pierre America (ed.), Proc. Object-Oriented Programming, 5th European Conference (ECOOP'91), *Springer Verlag Lecture Notes in Computer Science* **512** 133–147.

Willian A. Howard (1980). The formulae-as-types notion of construction. In: James R. Hindley and Jonathan P. Seldin (eds.), *To H. B. Curry: Essays on combinatory logic, lambda calculus and formalism*, Academic Press.

Cliff B. Jones (1983). Specification and design of (parallel) programs. In: R. E. A. Mason (ed.), *Information Processing 83: Proc. 9th IFIP World Computer Congress*, Elsevier 321–332.

Cliff B. Jones (1990). *Systematic Software Development Using VDM.* Prentice Hall, 2nd edition.

Bengt Jonsson and Yih-Kuen Tsay (1995). Assumption/guarantee specifications in linear time temporal logic. In: Peter D. Mosses, Mogens Nielsen, and Michael I. Schwartzbach (eds.), Theory and Practice of Software Development (TAPSOFT'95), *Springer-Verlag Lecture Notes in Computer Science* **915** 262–276.

Ragui F. Kamel (1987). Effect of Modularity on System Evolution. *IEEE Software* **4** (1) 48–54.

Ron Koymans (1987). Specifying message passing systems requires extending temporal logic. In: *6th ACM Symposium on Principles of Distributed Computing*, ACM Press 191–204.

Jeff Kramer and Jeff Magee (1990). The evolving philosophers problem: Dynamic change management. *IEEE Transactions on Software Engineering* **16** (11) 1293–1306.

Fred Kröger (1987). *Temporal Logic of Programs*, *EATCS Monographs on Theoretical Computer Science* **8**. Springer-Verlag.

Fred Kröger (1990). On the interpretability of arithmetic in temporal logic. *Theoretical Computer Science* **73** (1) 47–60.

Derek Lam, Jan Jannink, Donald C. Cox, and Jennifer Widom (1996). Modelling location management in personal communication systems. In: *Proc. of International Conference on Universal Personal Communications (ICUPC'96)*, IEEE Press 596–601.

Leslie Lamport (1994). The temporal logic of actions. *ACM Transactions on Programming Languages and Systems* **16** (3) 872–923.

Manny M. Lehman and Lazlo Belady (1985). *Program Evolution — Process of Software Change.* Academic Press.

Manny M. Lehman, Vic Stenning, and Wladislaw Turski (1984). Another look at software design methodology. *ACM SIGSOFT Software Engineering Notes* **9** (2) 38–53.

Ulf Leonhardt and Jeff Magee (1996). Towards a general location service for mobile environments. In: *Proc. 3rd International Workshop on Service in Distributed and Networked Environments (SDNE'96)*, IEEE Press 43–50.

Karl J. Lieberherr, Ignacio Silva-Lepe, and Cum Xiao (1994). Adaptive object-oriented programming using graph-based customisation. *Communications of the ACM* **37** (5) 94–101.

Barbara H. Liskov and Stephen N. Zilles (1975). Specification techniques for data abstractions. *IEEE Transactions on Software Engineering* **1** (1) 7–19.

Zhiming Liu and Mathai Joseph (1992). Transformation of programs for fault-tolerance. *Formal Aspects of Computing* **4** 442–469.

Pattie Maes (1987). Concepts and experiments in computational reflection. In: *Proc. Object-Oriented Programming Systems, Languages and Applications (OOPSLA'87)*, ACM Press 147–155.

Tom Maibaum and Wladyslaw Turski (1984). On what exactly is going on when software is developed step-by-step. In: *Proc. 7th International Conference on Software Engineering (ICSE'84)*, IEEE Computer Society Press 525–533.

Tom Maibaum, Martin Sadler, and Paulo Veloso (1984). Logical specification and implementation. In: Mathai Joseph and Rudrapatna Shyamasundar (eds.), Proc. 4th Conference on Foundations of Software Technology and Theoretical Computer Science, *Springer-Verlag Lecture Notes in Computer Science* **181** 13–30.

Tom Maibaum, Paulo A. S. Veloso, and Martin Sadler (1985). A theory of abstract data types for program development: Bridging the gap? In: Hartmut Ehrig *et al.* (eds.), Proc. International Conference on Theory and Practice of Software Development (TAPSOFT'85), vol. II, *Springer Verlag Lecture Notes in Computer Science* **185** 214–230.

Zohar Manna and Amir Pnueli (1979). The modal logic of programs. In: Hermann A. Maurer (ed.), Proc. 6th International Symposium on Automata, Languages and Programming (ICALP'79), *Springer-Verlag Lecture Notes in Computer Science* **71** 385–410.

Zohar Manna and Amir Pnueli (1983). How to cook a temporal proof system for your pet language. In: *Proc. 10th Symposium on Principles of Programming Languages*, ACM Press 141–154.

Zohar Manna and Amil Pnueli (1989). The anchored version of the temporal framework. In: Jaco W. de Bakker, Willem-Paul de Roever, and Grzegorz Rozenberg (eds.), Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, *Springer-Verlag Lecture Notes in Computer Science* **354** 200–284.

Satoshi Matsuoka (1993). *Language Features for Re-use and Extensibility in Concurrent Object-Oriented Programming.* PhD thesis, University of Tokyo, Japan.

Peter J. McCann and Gruia-Catalin Roman (1998). Compositional programming abstractions for mobile computing. *IEEE Transactions on Software Engineering* **24** (2) 97–110.

José Meseguer and Narciso Martí-Oliet (1995). From abstract data types to logical frameworks. In: Egidio Astesiano, Gianna Reggio, and Andrzej Tarlecki (eds.), Recent Trends in Data Type Specificaiton: 10th Workshop on Specification of Abstract Data Types, *Springer-Verlag Lecture Notes in Computer Science* **906** 48–80.

José Meseguer (1989). General logics. In: Hans Dieter Ebbinghaus *et al.* (eds.), *Logic Colloquium 87*, North Holland 275–329.

José Meseguer (1990). A logical theory of concurrent objects. In: Jerry L. Achibald and K. C. Burgess Yakemovic (eds.), *Proc. Object-Oriented Programming, 4th European Conference and Object-Oriented Programming Systems, Languages and Applications (ECOOP/OOPSLA'90)*, ACM Press 101–115.

José Meseguer (1992). Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science* **96** (1) 73–155.

Robin Milner, Joachim Parrow, and David Walker (1992). A calculus of mobile processes, I and II. *Information and Computation* **100** (1) 1–40 and 41–77.

Robin Milner, Joachim Parrow, and David Walker (1993). Modal logics for mobile processes. *Theoretical Computer Science* **114** (1) 149–171.

Robin Milner (1980). *A Calculus of Communicating Systems*, *Lecture Notes in Computer Science* **92**. Springer-Verlag.

Robin Milner (1983). Calculi for synchrony and asynchrony. *Theoretical Computer Science* **25** 267–310.

Robin Milner (1989). *Communication and Concurrency.* Prentice-Hall.

Robin Milner (1996). Semantic ideas in computing. In: Ian Wand and Robin Milner (eds.), *Computing Tomorrow: Future Directions in Computer Science*, Cambridge University Press 246–283.

Fredrik Orava and Joachim Parrow (1992). An algebraic verification of a mobile network. *Formal Aspects of Computing* **4** 497–543.

Susan Owicki and David Gries (1976). An axiomatic proof technique for parallel programs. *Acta Informatica* **6** 319–340.

Susan Owicki and Leslie Lamport (1982). Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems* **4** (3) 455–495.

Paritosh K. Pandya and Mathai Joseph (1991). P-A logic: a compositional proof system for distributed systems. *Distributed Computing* **5** 37–54.

Francesco Parisi-Presicce and Alfonso Pierantonio (1994). An algebraic theory of class specification. *ACM Transactions on Software Engineering and Methodology* **3** (2) 166–199.

David L. Parnas (1978). Designing software for ease of extension and contraction. In: *Proc. 3rd International Conference on Software Engineering (ICSE'78)*, IEEE Computer Society Press 264–277.

Lawrence Paulson (1994). *Isabelle: A generic theorem prover*, *Lecture Notes in Computer Science* **828**. Springer-Verlag.

Amir Pnueli (1977). The temporal logic of programs. In: *Proc. 18th Symposium of Foundations of Computer Science*, IEEE Press 45–57.

Amir Pnueli (1985). In transition from global to modular temporal reasoning about programs. In: Krzystof R. Apt (ed.), Logics and Models of Concurrent Systems, *NATO ASI Series* **13**, *Springer -Verlag* 123–144.

Amir Pnueli (1985). Linear and branching structures in the semantics and logics of reactive systems. In: Wilfried Brauer (ed.), Proc. 12th International Colloquium on Automata, Languages and Programming, *Springer-Verlag Lecture Notes in Computer Science* **194** 15–32.

Michel Raynal and Masaaki Mizzymo (1993). How to find his way in this jungle of consistency criteria for distributed shared memories. In: *Proc. 4th Workshop on Future Trends of Distributed Computing Systems*, IEEE Computer Society Press 340–346.

Gruia-Catalin Roman, Peter J. McCann, and Jerome Y. Plun (1997). Mobile UNITY: reasoning and specificaton in mobile computing. *ACM Transactions on Software Engineering and Methodology* **6** (3) 250–282.

Mark Ryan, José L. Fiadeiro, and Tom Maibaum (1991). Sharing actions and attributes in modal action logic. In: Takayasu Ito and Albert Meyer (eds.), Theoretical Aspects of Computer Software, *Springer Verlag Lecture Notes in Computer Science* **526** 569–593.

Vladmir V. Rybakov (1997). *Admissibility of Logical Inference Rules*, *Studies in Logic and the Foundations of Mathematics* **136**. Elsevier.

Czesław Ryll-Nardzwski (1952). The role of the axiom of induction in elementary mathematics. *Fundamenta Mathematicae* **39** 239–263.

Motoshi Saeki, Takeshi Hiroi, and Takanoru Ugai (1993). Reflective specification: Applying a reflective language to formal specifications. In: *Proc. 7th International Conference on Software Specification and Design*, IEEE Computer Society Press 204–213.

Hanan Samet (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys* **2** (2) 187–260.

Beverly Sanders, Berna L. Massingill, and Svetlana Kryukova (1997). Specification and proof of and algorithm for location management for mobile communication devices. In: Dominique Mery (ed.), *Proc. 2nd. International Workshop on Formal Methods for Parallel Programming: Theory and Applications*, 9–23.

Krister Segeberg (1970). Modal logic with linear alternative relations. *Theoria* **36** 301–322.

Amílcar Sernadas, Cristina Sernadas, and José Félix Costa (1995). Object specification logic. *Journal of Logic and Computation* **5** (5) 603–630.

Shaul Simhi, Vered Gafni, and Amiram Yehudai (1996). Combining reflection and finite state diagrams for design enforcement. *Theory and Practice of Object Systems* **2** (4) 269–282.

A. Prashad Sistla, Emerson M. Clarke, Nissim Francez, and Albert R. Meyer (1984). Can message buffers be axiomatised in linear temporal logic? *Information and Computation* **63** (1–2) 88–112.

J. Michael Spivey (1989). *The Z Notation: A Reference Manual. International Series in Computer Science.* Prentice-Hall.

Mike Spreitzer and Marvin Theimer (1994). Architectural considerations for scalable, secure, mobile computing with location information. In: *Proc.*

*14th International Conference on Distributed Computing Systems*, IEEE Computer Society Press 29–38.

Richard M. Stallman (1981). EMACS: The extensible, customisable self-documenting display editor. AI memo 519a, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Colin Stirling (1992). Modal and temporal logics. In: Sampson Abramsky, Dov Gabbay, and Tom Maibaum (eds.), Volume II, *Oxford Science Publications Handbook of Logic in Computer Science* 477–563.

Daniel Sturman and Gul Agha (1995). A protocol description language for customising failure semantics. In: *Proc. 13th Symposium on Reliable Distributed Systems*, IEEE Computer Society Press 148–157.

Manfred Egon Szabo (ed.) (1969). *The Collected Papers of Gerhard Gentzen.* North-Holland.

Andrzej Szałas (1988). Towards the temporal approach to abstract data types. *Fundamenta Informaticae* **11** 49–64.

Yasayuki Tahara *et al.* (1996). An algebraic semantics of reflective objects. In: Kokichi Futatsugi and Satoshi Matsuoka (eds.), Proc. 2nd International Symposium on Object Technologies for Advanced Software (ISOTAS'96), *Springer-Verlag Lecture Notes in Computer Science* **1049** 173–189.

Carolyn Talcott (1996). An actor rewriting theory. In: 1st International Workshop on Rewriting Logic and its Applications, *North Holland Electronic Notes in Theoretical Computer Science* **4** 360–383.

Carolyn Talcott (1996). Interaction semantics for components of distributed systems. In: Elie Najm and Jean-Bernard Stefani (eds.), *Proc. 1st IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'96)*, Chapman and Hall.

Carolyn Talcott (1997). Composable semantic models for actor theories. In: Martín Abadi and Takayasu Ito (eds.), Theoretical Aspects of Computer Science: 3rd International Symposium (TACS'97), *Springer Verlag Lecture Notes in Computer Science* **1281** 321–364.

Bent Thomsen (1991). *Calculi for Higher-Order Communicating Systems.* PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine, London, UK.

Mario Tokoro (1993). The society of objects. In: Jerry L. Achibald and Mark C. Wilkes (eds.), *Adendum to the Proc. Object Oriented Programming Systems, Languages and Applications (OOPSLA'93)*, ACM Press.

Anne S. Troelstra and Helmut Schwichtenberg (1996). *Basic Proof Theory.* Cambridge University Press.

Wladislaw Turski and Tom Maibaum (1987). *The Specification of Computer Programs.* Addison Wesley.

Johan van Benthem (1984). Correspondence theory. In: Dov Gabbay and Franz Guentener (eds.), *Handbook of Philosophical Logic*, volume II, D. Reidel 167–248.

Dirk van Dalen (1994). *Logic and Structure.* Springer Verlag, 3rd edition.

Jean van Heijenoort (ed.) (1967). *From Frege to Gödel: A source Book in Mathematical Logic.* Harvard University Press, 3rd edition.

Paulo A. S. Veloso (1992). Yet another cautionary note on conservative extensions: A simple case with a computing flavour. *Bulletin of the European Association for Theoretical Computer Science* **46** 186–192.

Nalini Venkatasubramanian and Carolyn Talcott (1993). A meta architecture for distributed resource management. In: *Proc. Hawaii International Conference on System Sciences*, IEEE Computer Society Press 124–133.

Mitchell Wand and Daniel P. Friedman (1988). The mystery of the tower revealed: A non-reflective description of the reflective tower. In: Pattie Maes and Daniela Nardi (eds.), *Meta-Level Architecture and Reflection*, North Holland 111–123.

Peter Wegner (1987). Dimensions of object-based language design. In: *Proc. Object Oriented Programming Systems, Languages and Applications (OOPSLA'87)*, ACM Press 168–182.

Roel J. Wieringa, Wiebren de Jonge, and Paul Spruit (1995). Using dynamic classes and role classes to model object migration. *Theory and Practice of Object Systems* **1** (1) 61–83.

C. Donald Wilcox and Gruia-Catalin Roman (1996). Reasoning about places, times and actions in the presence of mobility. *IEEE Transactions on Software Engineering* **22** (4) 225–247.

Alberto Zanardo and José Carmo (1993). Ockhamist computational logic: Past-sensitive necessitation for CTL$^*$. *Journal of Logic and Computation* **3** (3) 249–268.

Alberto Zanardo (1996). Branching-time logic with quantification over branches: The point of view of modal logic. *Journal of Symbolic Logic* **61** (1) 1–39.

# Notation Index

# Subject Index