Symbolic Techniques for Value-passing Calculi

Julian Rathke

UNIVERSITY OF

SUSSEX

AT BRIGHTON

Submitted for the degree of D. Phil.

University of Sussex

March 1997

Preface

The contents of Chapters 3 and 4 are the result of collaborative research with my supervisor, Matthew Hennessy. Preliminary versions of these two chapters have been published as Sussex University technical reports [47, 48] and appear as an extended abstract in [46]. An earlier version of work in Chapter 5 has also appeared as a Sussex University technical report [91] written jointly with Matthew Hennessy. However, in its present form, Chapter 5 is my own work. An extended abstract detailing the contents of Chapter 6 is due to be published later this year [90].

Symbolic Techniques for Value-passing Calculi

Julian Rathke

Abstract

We investigate the use of symbolic operational semantics for value-passing process languages. Symbolic semantics provide analytical tools for reasoning about particular infinite state systems where traditional methods fail. We eschew the use of Milner's encoding of value-passing agents into pure process algebra and advocate the treatment of value-passing terms as first-order processes proper. Such an approach enables us to build finitary proof systems for reasoning within a variety of value-passing calculi. All work carried out here is parametric with respect to the language of data expressions and, as such, reasoning about processes must be done relative to reasoning about data.

Firstly, we consider the broadcast calculus CBS. A novel semantic notion of strong equivalence is derived using the technique of barbed bisimulations. We characterise this equivalence, equationally, using an infinitary rule of inference and then, by means of a symbolic semantics for CBS, demonstrate how this infinitary rule can be avoided. A similar programme of study is executed for the corresponding weak equivalence.

Secondly, the problem of model checking value-passing processes is addressed. We introduce a suitable specification language based on the modal μ-calculus and, using symbolic graphs as the underlying models, develop a sound tableaux based model checker for formulae in this new logic. We investigate the power of this model checker by proving relative completeness and incompleteness results for restricted classes of process and various sub-logics.

Finally, we pursue the technique of unique fixpoint induction in the setting of value-passing process languages. We offer an intuitive generalisation of unique fixpoint induction and show that it is derivable from existing formulations. Further to this, using symbolic techniques we present relatively complete proof systems for regular, guarded, value passing processes, which characterise both strong and weak bisimulation. A relationship between parameterised and unparameterised processes with parallel compositions is also described.

Submitted for the degree of D. Phil.

University of Sussex

March 1997

Contents

List of Figures

Chapter 1

Introduction

Communication and concurrency are the two fundamental concepts which are used to model and understand complex dynamic systems. A large system can be realised as a collection of many parts and, because autonomous agents go about their business independently, many isolated agents acting concurrently do not model systems well. In particular, they fail to capture the unified behaviour of the system, viewed holistically. We have come to accept that this aspect of systems can be modelled well by interaction or communication. This is the basic tenet of concurrency theory, or process calculus, that has given rise to a large body of work over the last few decades.

The first theory of concurrency might be considered to be the theory of Petri nets, [82, 92, 83]. These models arose as a generalisation of automata in which actions or events may be performed concurrently, but what was somehow absent from the net model approach was an appreciation of the structure of large systems, in particular, the idea of small components interacting to form a whole. These considerations were to be recognized in later years in the wake of the study of processes. Seemingly independently, theories of processes were developed around the idea of communicating agents, performing atomic, indivisible, actions, [7, 50, 70]; the calculi presented in [6, 74, 51] are the fruits of extensive research into processes, [10, 15, 44, 69, 71, 80] for example. These calculi all provide syntactic descriptions of communicating agents and, in the so-called *pure* forms of these languages, communication is modelled solely by synchronisation so that the fact that actual data may be transmitted from one agent to another is abstracted away. This is a perfectly valid abstraction for theoretical purposes but for many specifications one wishes to retain certain fundamental aspects of communication. For example, one may wish to describe protocols where actual messages are being sent between agents and future behaviour depends upon the content of such messages. Languages which incorporate this feature are typically distinguished from the pure variants by referring to them as value-passing calculi. The semantics of such calculi are the topic of this thesis.

The theory of process calculi has focussed, to a large extent, on pure calculi. Furthermore, the treatment of value-passing calculi has been somewhat neglected due to the fact that value-passing can be encoded in pure calculi in a straightforward manner. In a value-passing calculus the atomic actions are usually of the form $a!v$, meaning *send the value v on channel name a*, or of the form $a?v$ meaning *receive the value v on a*. Typically we do not know which value we are about to receive so we have a construct $a?x.p(x)$, which binds a variable x in the *abstraction* (a function from values to processes), p. We would consider this to denote the process which could perform any $a?v$ action and then subsequently behave as $p(v)$. The encoding of this construct into a pure language utilises the fact that each of the value actions can be considered as a single abstract action, written as a triple, $(a,!,v)$ or $(a,?,v)$. Thus $a?x.p(x)$ can be described in the pure calculus as a summation

$$\sum_{v \in Val} (a, ?, v).p(v).$$

This is often referred to as Milner's encoding but such a treatment of value-passing is prevalent in CSP also.

Let us consider the ramifications of accepting this approach to the theory of languages with value-passing. Firstly, the intended interpretation of an action $c!v$ is that a piece of data, v, is being sent along a channel named c and, in order to receive this piece of data, another process must be listening on channel c. This other process need not be listening for any particular piece of data but it must simply be listening for *some* value. Performing the encoding shifts the emphasis from communication of data to synchronisation. The interpretation one might give Milner's encoding of such an exchange of data is that the piece of data v is still being sent along channel c but, in contrast, the second process must be listening for that particular value in order for communication to occur. It so happens that the encoding guarantees that if a process is listening for any one value on a channel c then it must be listening for every value on the same channel. The way that the encoding ensures this property is by offering a choice of actions. Thus the single atomic action of listening on a channel is transformed into a multitude of independent actions being offered; this transformation has considerable effects upon the basic structure of processes.

Secondly, there is also the more practical issue of infinite summation. Many find it unrealistic to allow an infinitary operator in a language and, in order to implement Milner's encoding, such a beast is necessary whenever the value domain is infinite. Also, much of the theory developed for the pure language is restricted to the case of finite summation and is therefore inapplicable to value-passing languages with infinite data domains. Therefore we find Milner's encoding wholly unsatisfactory as a means of modelling value-passing processes. It is analogous to trying to understand predicate logic by reducing quantifiers to (infinite) conjunctions and disjunctions in the propositional case. We prefer to adopt a first-order approach to value-passing processes.

The use of process calculi for systems or protocol specifications has been encouraged in no small part due to presence of constructs like conditionals for modelling the communication flow of actual data. The ability to express properties of processes with particular reference to data is a generous but extremely useful feature of specification calculi. It is surprising that, in the complementary world of process modelling, there has been little attempt to shift away from the perspective of synchronisation as the basic communication mechanism. Labelled transition systems [58] have long been used as a basic model of reactive computation in which a behaviour is not simply a function of input and output, but models responses to stimuli over time. Transition systems themselves are considered to give excessive detail when describing processes so the usual approach to studying transition system semantics is to construct a graph and then quotient the graph by some notion of equivalence. The three most notable equivalences used are

- trace equivalence (or language equivalence, as it is referred to in automata theory) [51],
- testing equivalence [39], and
- bisimulation equivalence, [80, 74].

Both trace and testing equivalence are coarser than bisimulation which is widely considered to be the finest equivalence one need study for transition systems. Roughly speaking, a bisimulation is a binary relation on processes such that if p and q are related and p can perform some action then q must also be able to perform the same action and the resulting states reached must also be related. Symmetrically, if q can perform some action then this must be matched by p. We say that p and q are bisimulation equivalent, or *bisimilar*, if there exists a bisimulation containing (p, q). Bisimulation has pleasing algebraic properties. It can be characterised equationally or by using a form of modal logic, [44] and it has also been shown that the theory of non-well founded sets is closely related to transition systems quotiented by bisimulation equivalence [2]. Bisimilarity is coinductively defined and proofs of equivalence by finding witnessing bisimulation relations provide an elegant example of coinductive reasoning. A measure of the success of bisimulation is that transition systems and bisimulation equivalence are now also being used as a convenient model of and coinductive proof technique for sequential computation in the functional setting [31, 32].

We have been describing what is referred to as *strong* bisimulation equivalence. There is in fact a much coarser notion of bisimulation equivalence called *weak* bisimulation. The distinguishing property of weak bisimulation is that communication is no longer treated as an observable action. In CCS this is codified by denoting the act of communication explicitly using a dummy symbol τ. Strong bisimulation takes τ actions into account and keeps track of internal computations whereas weak bisimulation treats this as unobservable and abstracts from them. We consider bisimulation based equivalences exclusively in this thesis and refer the reader to [52] for a treatment of the semantics of testing equivalence for value-passing languages.

One might argue that labelled transition systems are perfectly adequate to model the actual behaviour of systems. Their appropriateness seems to lie in their ability to capture the branching structure of processes, which is clearly the interesting part of process behaviour in the concurrent setting. However, in order to model a value-passing process as a labelled transition system we need to invoke Milner's ubiquitous encoding and the structural integrity of the process is compromised. A proposal for modelling value-passing processes directly in some sort of first-order transition system is found in [40]. Hennessy and Lin propose *symbolic graphs* as a suitable generalisation of transition systems.

We use the term *symbolic* to represent the fact that a syntactic approach to data is adopted in order to build our graphs. Thus, values and value-expressions are uninterpreted and are represented solely by their function symbols. With this in mind, one might consider *syntactic graphs* to be a acceptable alternative name for symbolic graphs. We prefer to use the term symbolic to reflect its usage in the work of Burch et al [17]. The more general use of the word *symbolic* in [17] refers to finding an appropriate representation of a problem's state-space, rather than working with the state-space explicitly. For example, using BDDs to represent transition relations and formulas for model checking in the modal μ-calculus, [23], is an instance of a symbolic approach. Our symbolic graphs provide a symbolic representation of the underlying graphs in the sense of Burch et al.

Symbolic graphs provide a means of retaining the commendable ability of transition systems to model the branching structure of processes and at the same time do not allow the detrimental effects of Milner's encoding of value-passing languages to impinge upon this structure. Although symbolic graphs carry an amount of syntactic baggage, and this is clearly an undesirable feature of basic models, we see that the actual syntax used in the graphs relates to data only. All of the real process information is modelled exactly as one would model a pure process in a labelled transition system.

We demonstrate how symbolic graphs capture structures of processes which evade transition systems with the following example. Consider a simple value domain containing only the values 0 and 1. Let p be the process $(c?x.(x=0) \rightarrow p') + (c?x.(x=1) \rightarrow p')$ which can receive either value and, depending on the value received, deadlock or continue to behave like p' with x substituted by the appropriate value. The choice means that, when offered a communication, the process will decide whether to use the left or the right summand and the liveness of the process then depends upon the value received. In contrast to this we let q be the process $c?y.p' + c?y.\mathbf{O}$ which again can receive either of the values 0 or 1 on the channel c and then deadlock or continue to behave as p'. With q however, after the choice whether to use the left or right summand is made, there is no dependency upon the data to determine whether deadlock occurs. This subtlety is not captured by the transition system models of p and q:

$$\begin{tikzcd}
 & p'[0/x]\ldots & & & p'[0/x]\ldots \\
 & \bullet & & & p'[1/x]\ldots \\
p \arrow[ruu, "c?0"] \arrow[ru, "c?1"'] \arrow[rd, "c?0"] \arrow[rdd, "c?1"'] & & & q \arrow[ruu, "c?0"] \arrow[ru, "c?1"'] \arrow[rd, "c?0"] \arrow[rdd, "c?1"'] & \\
 & \bullet & & & \bullet \\
 & p'[1/x]\ldots & & & \bullet
\end{tikzcd}$$

It is clear to see that the transition system models of p and q are identical. The standard notion of bisimulation for value-passing processes that one obtains via Milner's encoding into pure languages, and adopting Park's original notion of bisimulation [80] for the resulting transition systems, is called *early* bisimulation [75]. It is no surprise that p and q are deemed to be early bisimilar. There is an acceptable alternative definition of bisimulation for value-passing languages called *late* bisimulation [75]. This equivalence is more sensitive to branching structures than early bisimulation and consequently would distinguish p and q. In order to obtain late bisimulation as an instance of Park's definition, a modification of the graphs for p and q is required. That is, we must use a different semantics in order to study the finer equivalence. There seems to be no clear argument as to which of these equivalences is the more suitable and it depends largely on whether one believes late bisimulation to be just too fine.

The symbolic graph models of p and q take the readily distinguishable forms

$$\begin{tikzcd}
 & x=0 \rightarrow p'\ldots & & & p'\ldots \\
p \arrow[ru, "c?x"] \arrow[rd, "c?x"'] & & & p \arrow[ru, "c?x"] \arrow[rd, "c?x"'] & \\
 & x=1 \rightarrow p'\ldots & & & \bullet
\end{tikzcd}$$

and we see that symbolic semantics are discriminating enough to model late bisimulation. It is also possible to give a characterisation of early bisimulation using the *same* symbolic semantics. We see in the next chapter that the difference between early and late equivalence is shown clearly by symbolic semantics. We refer to these characterisations of bisimulation upon symbolic graphs as *symbolic* bisimulation.

It is our opinion that symbolic graphs provide a suitable semantics for modelling the structure of value-passing processes. It is said that the proof of the pudding is in the eating, so we must demonstrate that the symbolic approach is in fact useful and provides a better understanding of value-passing processes than that obtainable from the transition system models. A general philosophy behind the symbolic approach is that we are interested in process, but not data reasoning. This is reflected by the fact that symbolic semantics capture the structure of processes yet leave the syntax of data intact. In accordance with this policy we work parametrically in the data language, although, without a clear definition of a language of data, we cannot hope to provide total verifications of properties of value-passing processes. We therefore content ourselves with establishing partial verifications: on the assumption that certain properties of data hold we can verify properties of processes. In most cases our work involves reducing judgements about value-passing process terms down to judgements about data only. One of the benefits of this approach is that we can more easily identify the rôle of data in the description of a process. For instance, although it is known that bisimulation is decidable for various classes of infinite state transition systems [19, 20, 102], it is unknown whether bisimulation is decidable for regular value-passing processes, say, because such a question depends ultimately on the domain of values used and the expressivity of the data expressions allowed. The algorithms for partially deciding bisimulations of symbolic graphs in [40] entail the result that, given languages of data expressions and boolean conditions upon data such that validity in the first-order boolean language is decidable, then bisimulation for the class of regular processes yielding finite symbolic graphs, defined over that data language, is also decidable. Thus, any undecidability here would be due to data and not process behaviour.

Another area in which the symbolic approach has been useful is the area which we plumb in this thesis, *viz* the development of proof systems for value-passing processes. One can think of the nodes of a symbolic graph as representing an open term of a first-order process language. Thus we would use symbolic semantics to reason at the level of open terms. This is precisely the approach which is adopted in [41, 43], say, in which *finitary* proof systems are developed to reason about value-passing languages by working with open terms directly, rather than closed instantiations of open terms.

The notable feature of the proof systems in [41] and [43] is that they are proof systems for reasoning about open process terms and as such, judgements of these proof systems are made relative to boolean expressions. Thus, to characterise bisimulation we have equations of the form

$$\vdash b \rhd t = u$$

where b is some boolean predicate expression and t and u are open process terms. We should point out that throughout this thesis we will refer to boolean expressions drawn from two different languages. Firstly, there is the formal language of boolean predicates which can be used in the syntax of value-passing languages and symbolic graphs; we refer to this language as *BoolExp*. Secondly, we allow ourselves some freedom in using a metalanguage of boolean properties. This metalanguage is not specified precisely and we will liberally assume certain expressive powers here - *BoolExp* ought to be contained in the metalanguage at least. The boolean b in the sequent above will typically be expressed in the boolean metalanguage.

We can understand boolean guarded sequents as a collection of closed term sequents, one for each instantiation of the free variables of b such that b holds true. In each instantiation, δ, say, we instantiate the free variables of t and of u according to δ and the resulting sequent is $\vdash t\delta = u\delta$. The important point is that where we would have needed an infinitary proof rule to state that input prefixing preserves bisimulation,

$$\frac{\vdash t[v/x] = u[v/x] \quad \forall v \in Val}{\vdash c?x.t = c?x.u}$$

for closed processes $c?x.t = c?x.u$, the open term equivalent of this rule is simply that

$$\frac{\vdash b \rhd t = u}{\vdash b \rhd c?x.t = c?x.u}$$

provided that x does not occur freely in b.

Certain proof rules in [41] and [43] carry side-conditions involving data, for instance, the rule which states that output prefixing preserves bisimulation is

$$\frac{\vdash b \rhd t = u}{\vdash b \rhd c!e.t = c!e'.u} \quad \text{if } b \text{ implies } e = e'.$$

Typically these are statements about the boolean language of data expressions. We might think of these side-conditions as proof obligations which can be proved in an auxiliary proof system for reasoning with data alone. Thus the side-conditions would be some kind of interface to a proof system for data, so rather than becoming burdened with the intricacies of data reasoning we simply assume an oracle for deciding questions of data. All results such as soundness and completeness will rely on this oracle and will consequently be relative soundness and completeness results. It is the fact that the work is parametric with respect to data domains that demands that verifications be relative to data. Throughout the thesis we will occasionally use the terms soundness or completeness without the *relative* qualification.

In this thesis we study the benefits of symbolic semantics for developing proof systems for value-passing languages. In particular we concern ourselves with the specific problems of finding proof systems for characterising bisimulation equivalence for a language featuring broadcast communication, developing a modal specification logic and model checker for generic value-passing languages, and investigating the proof technique of unique fixpoint induction for value-passing CCS. We describe each of these topics in turn in the following three sections.

1.1 Value-passing by broadcast

Many process calculi, including CCS and ACP, use handshaking communication as a primitive notion. In contrast to this there are languages which prefer to adopt a multiway synchronisation operator such as Hoare's CSP, LINDA [18, 60] and ESTEREL [11, 12]. In [85, 86, 87] Prasad argues that multiway or broadcast communication is a more natural concurrency primitive than the handshake communication of CCS, say. The argument centres upon the use of broadcast communication in Local Area Networks and Ethernet, [68] and an analogy is drawn with human speech as a communication mechanism. He also argues that multiway synchronisation [84] is unsuitable for modelling broadcast communication because abstract actions are not typed as input or output actions; in a broadcast exchange there should be a unique sender with multiple recipients. Thus Prasad introduces his Calculus of Broadcasting Systems, CBS, as a variant of value-passing CCS which uses broadcast communication as a primitive in place of the familiar handshake.

The calculus CBS differs from CCS in the use of channel names: the implicit name matching, which is central to synchronisation in CCS, determines which agents in a network may communicate with each other so that there is a fixed network of communication. This unsatisfying monolithic view of networks led some to develop calculi for dynamic descriptions of network connections, the most notable of these being the π-calculus [75], whereas those using broadcast communication take the extremist view of considering totally dense networks. All agents may communicate with each other, rendering channel names obsolete, but this does not preclude passing messages in localised parts of the network. CBS provides a facility for modifying the data flowing in and out of an agent. This can be thought of as an encryption/decryption service which can make messages private by enciphering them in such a way that only the local agents can receive them. Preventing certain agents from receiving enciphered messages necessitates a form of pattern-matching on inputs. Where a CCS process listening for a value on some channel c is in fact always listening for all values, a CBS process listening for a value can be listening for some designated set of values. This is one feature of CBS that makes its theory of interest and demands a generalised form of symbolic semantics.

On the surface it would appear that there shouldn't be too much difference between the theory of value-passing CCS and CBS. So long as we restrict our attention to the finite sublanguages of each of these the difference in the communication mechanism is accounted for by using different expansion rules for parallelism. The reason the theory is actually significantly different is a result of the notion of observational equivalence which we use. In a handshaking calculus it seems quite reasonable to treat reception as an observable action. This is due to the fact that in order to receive some data the recipient has to make its presence known to the sender. However, when we move to a broadcasting calculus, observability of reception is not so obvious, because a process which has just transmitted a value has no real way of telling which other processes, if any, received that value. The difference in the notion of observable actions between a handshaking and a broadcasting calculus provide the differences in the theories of the calculi. We consider an original notion of bisimulation which we call *noisy* bisimulation and we justify our interest in this equivalence by appealing to *barbed* bisimulation, [76, 94].

Barbed bisimulation is advocated by Milner and Sangiorgi as a general purpose observational equivalence which is definable in most process languages. The idea is that one defines a bisimulation type relation, much in the same way as the standard definition, except that one considers how processes perform *computations* rather than actions. In CCS, computation is an internal τ action corresponding to the result of a synchronisation, whereas, in CBS, computation is the action of transmitting noise or indecipherable messages. The reason barbed bisimulation is described as an observational equivalence is that one typically studies the congruence obtained by considering barbed bisimulation in all contexts. Thus we observably test processes by situating them in contexts which induce them to perform computations.

1.2 Model checking

The equational approach to verification consists of specifying a process in some calculus, implementing this process in the same calculus and then showing that the two descriptions are equal according to some notion of equivalence for the calculus. An alternative to this approach is to specify properties of a process in some language which is distinct from the process language itself. We then define a satisfaction relation between processes and specifications and proceed to show that this relation holds for the particular process and property under examination. In particular, an instance of this approach is where the specification language is the modal μ-calculus, [88, 61]. The property logic is interpreted over the class of transition systems being used to model processes. Satisfaction can then be defined straightforwardly by saying that a process satisfies a formula if its representation as a transition system is present in the interpretation of the formula. We refer to this sort of validation as *model checking*.

There has been a great deal of activity investigating this approach to verification for pure process calculi and labelled transition systems, [21, 89, 22, 59, 24, 103, 107, 25, 23, 4]. Many of these are concerned with efficiency of the technique and concentrate on aspects such as taming the state explosion problem with Binary Decision Diagrams (BDDs) [16], and improving tableaux based algorithms. There has been much success in this area and a high level of automation has been achieved [17]. In addition to this we see that elegance and good structural reasoning underpin this method of property verification; papers such as [101, 103, 107] present insightful characterisations of the satisfaction relation using proof tableaux and games.

For the most part, research on model checking is with respect to finite models. Clearly, using BDD models and tableaux based checkers will be infeasible for value-passing processes and even with finite value-domains the state spaces can be so large as to make automation impractical. An investigation into model checking μ-calculus properties for arbitrary infinite state systems is carried out in [14] where a sound and complete model checker for arbitrary transition systems is presented. This model checker is indeed suitable for model checking value-passing processes (using Milner's encoding) but the completeness proof contained in [14] is necessarily non-constructive. This is due to the undecidability of satisfaction of μ-formulae over infinite state systems. What we are lacking is a constructive method for model checking μ-formulae for value-passing processes.

There has been some recent work also addressing the issue of model checking for value-passing processes in [38]. Here, a generalisation of the μ-calculus suitable for specification of value-passing processes is introduced. The logic proposed, independently, in [38] is very similar to our own property logic in terms of both syntax and expressive power. A compositional proof system for soundly establishing satisfaction is proposed although the power of this system is not yet determined. In a similar vein, modal logics for mobile processes have been developed in [27, 3] and sound and complete tableaux systems for deciding satisfaction are presented there.

1.3 Unique fixpoint induction

We briefly describe the proof method of unique fixpoint induction for equational reasoning with recursively defined processes. This is employed by Milner in [72, 73] and is based on the fact that declarations used for defining recursive processes yield unique behaviours. The property of bisimulation equivalence we exploit is that, given a declaration

$$X \Longleftarrow E$$

and two processes p and q such that p is bisimilar to $E[p/X]$ and q is bisimilar to $E[q/X]$, then p and q must themselves be bisimilar. This property does not hold for arbitrary process descriptions E, but is guaranteed for guarded descriptions where occurrences of X in E are within the scope of an action prefix. The same property holds for processes with respect to trace equivalence, [51]. It is shown in [74] that this proof technique alone, along with rules for removing unguardedness is sufficient for characterising bisimulation equivalence for regular process terms. This is unsurprising as the unique fixpoint induction proof rule seems to capture the coinductive flavour of bisimilarity perfectly. The principle can be expressed as an inference rule

$$\frac{\vdash p = E[p/X]}{\vdash p = X}$$

where $X \Longleftarrow E$ is guarded; it is the guardedness that ensures the soundness of this rule. A successful proof by unique fixpoint induction corresponds roughly to finding matching loops in the underlying transition graphs of processes and is somewhat analogous to reasoning by invariants.

With regards to value-passing calculi, the use of unique fixpoint induction is a recent development. It was anticipated that the technique could be integrated comfortably into the proof system of [41] and recent work [42] began this investigation. The use of unique fixpoint induction for reasoning in the π-calculus has also been studied in [64, 62].

1.4 Overview of the thesis

The aim of this thesis is to pursue the development of the symbolic approach to the semantics of value-passing languages and argue that substantial gains in the theory of processes can be obtained by dispensing with the traditional approach of encoding value-passing processes into pure algebras. We also wish to demonstrate the versatility of the symbolic approach by applying it to various problems concerning the verification of processes with data. As with any new proof technique, a solid theoretical foundation is essential for recognition as a useful and insightful tool for the analysis of programs. We hope that the work presented here will strengthen the position of symbolic semantics and at the same time provide a better understanding of the particular problems that we tackle. The overall structure and interdependencies of the chapters of the thesis are illustrated in Figure 1.1.

- In Chapter 2 we recall the definitions of transition systems and bisimulation and we also motivate the difference between early and late bisimulation for value-passing languages. A description of what one requires of a language of data expressions is presented and a category of symbolic graphs is described on top of the category of data languages. We describe a functor from symbolic graphs to labelled transition systems called *concretion*. Transition systems obtained via concretion are shown to be strongly bisimilar to standard transition system semantics. Symbolic bisimulation as an equivalence on symbolic graphs is defined and its relationship, via concretion, to bisimulation on labelled transition systems is established. Finally, a brief mention of a recent extension of symbolic graphs called symbolic graphs with assignment is also made.

- Our study of the calculus of broadcasting systems begins in Chapter 3 by focussing on the strong bisimulation equivalence relation. We describe the calculus in detail and argue, using the technique of barbed bisimulations, for the suitability of a novel type of bisimulation equivalence which we call *noisy* equivalence. We proceed to characterise this new equivalence equationally for the finite sublanguage of CBS. This characterisation is achieved in three stages. Firstly, we consider a finite sublanguage of CBS which is very close to finite value-passing CCS in that pattern-matching on inputs is removed. A sound and complete proof theoretic characterisation for this simple sublanguage is presented for closed terms. This proof system features infinitary inference rules but we obviate the need for these by developing an equivalent proof system for open terms using the symbolic technique. The facility of pattern-matching on inputs is reintroduced explicitly and we show how the theory which has already been developed can be modified for this case. Finally, finite CBS is incorporated in the theory via expansion theorems. We show that the broadcast operator can be reduced to interleaving non-determinism and, more surprisingly, that the translation functions of CBS can be simulated by syntactic manipulations on action prefixes.

- The content of Chapter 4 follows on directly from that of Chapter 3. Again, we study the use of the symbolic semantics for the broadcast calculus, CBS, but now we are concerned with the weak equivalence. Weak equivalence for CBS is a coarser notion of equivalence wherein the transmission of noise is unobservable. To an observer placed within a network, the transmission of noise can signify local activity which it is not a party to and can thus be thought of as private or internal communication much like τ actions in CCS. We adopt weak barbed bisimulations as the basis for our weak equivalence and discover that these do in fact coincide with a notion of weak bisimulation proposed by Prasad [86]. The proof systems of the previous chapter are extended with the necessary axioms for describing the abstraction of noise and we present soundness and completeness results for the larger proof system. A short discussion on why late semantics are inappropriate for CBS concludes the chapter.

- In Chapter 5 we address the problem of model checking for value-passing processes. A generalised, first-order, modal μ-calculus suitable for value-based specifications is presented such that this new logic is a direct extension of the modal logic without fixpoints first proposed in [43]. A standard interpretation of the logic is given using concreted symbolic graphs, a subclass of transition systems, as the models. We show that the logic fully characterises late bisimulation equivalence and a tableaux style proof system for local model checking is presented. The use of symbolic semantics is crucial to our model checker as it was designed to allow one to establish properties of open process terms. Another essential ingredient of our proof system is a generalisation of Winskel's tag set method, [107], for recording the unfolding of fixpoint formulae but, rather than simply using nodes of a transition system as the tag information, we move to using nodes of a symbolic graph. This corresponds to reasoning about open terms instead of closed process terms. Soundness of the proof system is established in a standard manner and we prove that the first-order μ-calculus is too expressive for our proof system to be relatively complete, even for finite symbolic graphs. The basic problem is that inductive properties of the data domain can be described by least fixpoint formulae and our proof system does not yet allow induction over data. However, we show that, if one considers the sub-logic without least fixpoint formulae, we do in fact have a relatively complete model checker for finite symbolic graphs. Rather than dispensing with least fixpoint formulae entirely we also consider a different sub-logic which uses a restricted form of parameterisation. We argue that by reinterpreting this sub-logic symbolically we can show relative completeness for finite symbolic graphs. An example proof is presented at the end of the chapter.

- The power of unique fixpoint induction for value-passing processes is examined in Chapter 6. Recent research in this area [42] shows that the proof systems of [41] can be naturally extended with unique fixpoint induction and that the resulting system is sound and relatively complete with respect to strong bisimulation equivalence. The completeness result holds for a restricted class of regular processes. In particular, the use of parameterisation is heavily restricted in that parameters are assumed to be vectors of names alone. We show how an

$$\begin{tikzcd}
 & Ch.7 \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
Ch.4 \arrow[d, dash] & Ch.5 \arrow[dd, dash] & Ch.6 \arrow[ddl, dash] \\
Ch.3 \arrow[dr, dash] & & \\
 & Ch.2 \arrow[d, dash] & \\
 & Ch.1 &
\end{tikzcd}$$

Figure 1.1. Chapter dependencies

intuitive generalisation of the unique fixpoint induction rule, which is derivable from Hennessy and Lin's proposed rule, can be used to lift the restriction on parameters. We show relative completeness with respect to strong bisimulation for guarded regular processes. Extending this work further, we go on to characterise observation congruence and discover that the familiar τ laws due to Milner [74] can still be used to abstract from internal actions. A discussion on the relationship between parameterisation and parallel composition for value-passing languages is given and we conclude the chapter with an example equivalence proof.

- We end the thesis with a short chapter stating our conclusions and avenues for future research.

1.4.1 Pre-requisites

Although we review the basic definitions of transition systems, bisimulation and value-passing CCS, familiarity with pure process calculi would be a distinct advantage in reading this thesis. We refer the reader to the textbooks [74, 51, 6] for a good introduction to the subject. Issues relating to value-passing semantics are explained in full and no prior experience with such languages is required. The property logic, first-order μ-calculus, presented in Chapter 5 is based on the modal μ-calculus due to Kozen and Pratt [61, 88], and acquaintance with the use of modal languages to specify properties of processes would be useful; we refer the reader to the handbook chapter by Stirling [99].

Chapter 2

Symbolic Graphs and Symbolic Bisimulation

2.1 Transition systems

We recall the basic definition of labelled transition systems and refer the reader to [108] for further details of their categorical structure. A transition system is essentially a set of states with a relation, called the transition relation, which describes how the system may progress from one state to the next. A part of this description is to designate an *event* which is observed whilst doing this progression.

A ***labelled transition system*** is a triple $(N, \mathcal{L}, \longrightarrow)$ where

- N is a set of states, or nodes,
- $\mathcal{L}$ is a set of labels and
- $\longrightarrow \subseteq N \times \mathcal{L} \times N$ is the transition relation.

We will write $n \xrightarrow{a} n'$ to denote $(n, a, n') \in \longrightarrow$.

Given two transition systems $T_1 = (N_1, \mathcal{L}_1, \longrightarrow_1)$ and $T_2 = (N_2, \mathcal{L}_2, \longrightarrow_2)$, a morphism of these, $f : T_1 \to T_2$, is a pair (σ, λ) where $\sigma : N_1 \to N_2$ is a function on the nodes and $\lambda : \mathcal{L}_1 \rightharpoonup \mathcal{L}_2$ is a partial function on labels such that whenever $n \xrightarrow{a}_1 n'$ then either $\lambda(a)$ is undefined and $\sigma(n) = \sigma(n')$ or $\sigma(n) \xrightarrow{\lambda(a)}_2 \sigma(n')$.

Transition systems form a category **TS** by taking transition systems as objects with the morphisms as above, where composition is given by pointwise composition of functions and identities are pairs of identity functions.

2.2 Data domains

It is evident that the description of any message-passing, or value-based, process language is parameterised by a language for describing the data which are being communicated. This data language will be explicit in both the syntax and semantics of the process language. The philosophy of our work, and indeed that of the symbolic approach in general, is that a precise exposition of any data domain is unnecessary and the approach we advocate is to divorce reasoning about data from reasoning about processes as much as possible. We do not concern ourselves with the former and concentrate entirely on the latter. Our method of establishing properties of value-based processes is therefore *partial*, in that we reduce such properties to properties of the data domain only.

To this end we wish to impose as few restrictions on the language as possible and reason parametrically. Previous presentations of symbolic graphs simply assume certain properties of data languages to hold. We prefer to explicitly state the conditions a data domain is required to satisfy. Essentially these just say that we have identified sets of variables and values and we have signatures of expressions incorporating these which support substitution and evaluation.

A ***data language*** D is a quadruple (Var, Σ_V, I, Pr) such that the following conditions apply.

- Var is a countable set of variables.
- Σ_V is a signature of value expressions. We let Val denote the set of 0-arity functions of Σ_V. Let Σ_{V^+} denote the signature obtained from Σ_V by adding the constants Var and let $ValExp$ be the (initial) term algebra of this extended signature.
- I is a Σ_V-algebra with carrier Val such that $v_I = v$ for each $v \in Val$.
- Pr is a set of predicates, Q and functions Q_I which interpret these Q. That is, if Q has arity n then $Q_I : Val^n \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$.

Given such data we can define a function fv which returns the set of variables of a term of $ValExp$ by describing fv as the unique Σ_{V^+}-homomorphism from $ValExp$ to the Σ_{V^+}-algebra A, which has carrier $\mathcal{P}Var$ (where $\mathcal{P}$ denotes powerset) and

- $v_A = \emptyset$ for each $v \in Val$.
- $x_A = \{x\}$ for each $x \in Var$
- $f_A(S_1, \ldots, S_n) = \bigcup S_i$ for all other functions $f \in \Sigma_V$.

A term $e \in ValExp$ such that $fv(e) = \emptyset$ is called ***closed***. Note that the expressions in the term algebra of Σ_V are the closed terms of $ValExp$. We define an evaluation function $[\![\;]\!]$ on closed terms therefore as the unique Σ_V-homomorphism from the term algebra of Σ_V into the Σ_V-algebra I.

We define the boolean language $BoolExp$ to be first-order predicate logic with predicates given by the set Pr. For each $Q \in Pr$ of arity n and for all expressions $e_1, \ldots, e_n$ in $ValExp$ we have the predicate $Q(e_1, \ldots, e_n)$ in $BoolExp$. The variables of this predicate are simply the union of the variables of the e_i and we can extend the definition of $[\![\;]\!]$ to closed boolean expressions by simply interpreting the closed predicates $Q(e_1, \ldots, e_n)$ as $Q_I([\![e_1]\!], \ldots, [\![e_n]\!])$ and interpreting the connectives of first-order logic in the usual manner.

We notice that evaluation is restricted to closed expressions only. The operation of substitution in both $ValExp$ and $BoolExp$ is well understood so in order to evaluate an arbitrary expression, e or b say, we can substitute closed expressions for each of its (free) variables. It suffices to substitute values rather than closed expressions. By instantiating the free variables to values we provide an environment in which to interpret the arbitrary expression e or b. We describe such environments using functions: a function $\delta : Var \rightarrow Val$ is a ***data environment*** or simply an ***environment***. We define $[\![e]\!]\delta \in Val$ to be $[\![e[\tilde{v}/\tilde{x}]]\!]$ where $\tilde{v} = \delta(\tilde{x})$ and $\tilde{x} = fv(e)$. Similarly, for $b \in BoolExp$ we define $[\![b]\!]\delta$ to be $[\![b[\tilde{v}/\tilde{x}]]\!]$ and write $\delta \models b$ iff $[\![b]\!]\delta = \mathbf{tt}$. Formally, δ extends the Σ_V-algebra I to a Σ_{V^+}-algebra with carrier Val and $[\![\;]\!]\delta$ is the unique Σ_{V^+}-homomorphism from the term algebra to this extended algebra. We will often want to consider data environments which have been modified by some substitution, that is given δ and a substitution $[e/x]$ we describe an updated environment which behaves exactly as δ except that it maps the variable x to $[\![e]\!]\delta$; this environment will be notated $\delta[e/x]$. Note the following simple property of $BoolExp$:

$$\delta \models b[e/x] \iff \delta[e/x] \models b$$

which follows from the standard Substitution Lemma of predicate logic.

2.2.1 Morphisms of data languages

Given two data languages $D = (Var, \Sigma_V, I, Pr)$ and $D' = (Var', \Sigma_{V'}, I', Pr')$ it makes sense to ask what a map between these consists of. We allow a change of variable names so we have a bijection $d_V : Var \to Var'$ and we have an arity preserving map $d_\Sigma : \Sigma_V \to \Sigma_{V'}$. We notice that any $\Sigma_{V'}$-algebra A, may be considered to be a Σ_V-algebra by interpreting $f \in \Sigma_V$ as the interpretation of $d_\Sigma(f)$ in A. In particular this applies to the term algebra of $\Sigma_{V'}$ and the algebra I'. Thus we induce two Σ_V-homomorphisms, d^*_Σ being the map between the term algebras and $d^I_\Sigma : I \to I'$. The latter of these is obtained by considering the restriction of d_Σ to Val. We will also require an arity preserving map $d_P : Pr \to Pr'$ such that the interpretation of $d_P(Q)$ is precisely $d_P(Q_I)$.

So, a morphism $d : D \to D'$ is a triple (d_V, d_Σ, d_P). Now that we have a notion of morphism we can describe the category of data domains, *Data*: Objects are data domains and morphisms are the data domain morphisms with pointwise composition.

Proposition 2.2.1 *Data is a category.*

2.2.2 Examples of data domains

We give a few examples of data domains which we will have occasion to use throughout the thesis.

The natural numbers $\mathcal{N}$ are the example we use most frequently. We assume a countably infinite set of variables $Var = \{x_1, x_2, \ldots\}$ and have a signature $(\mathcal{N}, +, -, \times, \div, mod, div)$. The interpretation algebra is the obvious algebra of the arithmetic operations listed and we have predicates $>$ and $=$.

The one-point value domain 1 has a single variable x and a signature containing only the constant $*$. The interpretation I must be the singleton set $\{*\}$ and we have no predicates.

2.2.3 More about data environments

Given an environment $\delta : Var \to Val$ for a data domain D and a morphism $d : D \to D'$ we can induce an environment δ_d on D'. We define δ_d to be the unique environment which makes the following diagram commute.

$$\begin{tikzcd} Var \arrow[r, "d_V"] \arrow[d, "\delta"'] & Var' \arrow[d, "\delta_d"] \\ Val \arrow[r, "d_\Sigma"] & Val' \end{tikzcd}$$

A unique environment exists by virtue of the fact that d_V is a bijection. We write $b \models_d b'$ with $b \in BoolExp$ and $b' \in BoolExp'$ if for each δ such that $\delta \models b$ we have $\delta_d \models' b'$. This is a transitive relation with respect to composing data morphisms and coincides with the usual notion of implication when d is the identity map.

2.3 Symbolic graphs

We are now in a position to describe what Hennessy and Lin's original symbolic graphs consist of. Recall that they are intended as a generalisation of labelled transition systems and therefore can be loosely described as *a collection of nodes with an edge labelling relation.* More formally, a ***symbolic graph*** $\mathcal{G}$, is a quintuple $(D, N, Ch, N_\tau, \longmapsto)$ where

- D is a data domain.
- N is a set of nodes for which each element n, has an associated set of *free variables* denoted by $fv(n)$. We have $fv(n) \subseteq Var$.
- Ch is a set of channel names.

- N_τ is a set of neutral or internal actions.
- $\longmapsto : N \times BoolExp \times Act \times N$ is the edge labelling relation where

$$\begin{array}{rcl} Act & = & \{c?x \mid c \in Ch, x \in Var\} \\ & \cup & \{c!e \mid c \in Ch, e \in ValExp\} \\ & \cup & N_\tau. \end{array}$$

We write $n \overset{b,\alpha}{\longmapsto} n'$ if $(n, b, \alpha, n') \in \longmapsto$.

It is easy to lift the definition of free variables to actions by letting $fv(c?x) = fv(a) = \emptyset$ whenever $a \in N_\tau$ and $fv(c!e) = fv(e)$. We also define the complementary notion of *bound variable* of an action by $bv(c!e) = bv(a) = \emptyset$ and $bv(c?x) = \{x\}$.

Given these we ask that the following rationality conditions on free variables hold. If $n \overset{b,\alpha}{\longmapsto} n'$ then

$$fv(b) \cup fv(\alpha) \subseteq fv(n)$$

and

$$fv(n') \subseteq fv(n) \cup bv(\alpha).$$

We give an example symbolic graph in an obvious graphical notation. The free variable sets associated with the nodes are not depicted but we can assume that $fv(p) = \emptyset$ and $fv(q) = fv(r) = \{x\}$.

$$\begin{tikzcd} p \arrow[r, "\mathbf{tt}\text{, } c?x"] & q \arrow[r, bend left, "even(x)\text{, } c!x{+}1"] \arrow[r, bend right, "odd(x)\text{, } c!x"'] & r \arrow[l, "\mathbf{tt}\text{, } c?x"'] \end{tikzcd}$$

One might imagine the graph depicted describes a process p which can always receive a value on channel c and then output that same value if it is odd, or one greater if it is even. The node r has the same outgoing edges as p, in fact the nodes only differ in their free variable sets.

2.3.1 Morphisms of symbolic graphs

Symbolic graphs are generalisations of labelled transition systems and as such should have a similar notion of graph morphism. As yet there has been no attempt to describe these morphisms. Recall that a morphism of transition systems consists of two functions, one between the sets of nodes and the other, partial function, between the sets of labels. Similarly, a morphism of symbolic graphs will consist of a function between nodes of the graphs in question and a *function* between the sets of actions. However, since the sets of actions are built from Ch, N_τ and D, we ask for functions for each of these.

Given symbolic graphs $\mathcal{G} = (D, N, Ch, N_\tau, \longmapsto)$ and $\mathcal{G}' = (D', N', Ch', N'_\tau, \longmapsto')$ a morphism $f : \mathcal{G} \to \mathcal{G}'$ is a quadruple of maps $(\sigma, \lambda, \lambda_\tau, d)$ where

- $\sigma : N \to N'$
- $\lambda : Ch \rightharpoonup Ch'$ is a partial function.
- $\lambda_\tau : N_\tau \rightharpoonup N'_\tau$ is a partial function.
- $d : D \to D'$ is a data morphism.

The following conditions must hold on f: we require that the free variables of each node are preserved. This can be described easily using the powerset functor on **Set** by asking that the following diagram commutes.

$$\begin{tikzcd} N \arrow[r, "\sigma"] \arrow[d, "fv"'] & N' \arrow[d, "fv'"] \\ \mathcal{P}Var \arrow[r, "\mathcal{P}d_V"] & \mathcal{P}Var' \end{tikzcd}$$

Furthermore, we require that transitions be preserved. This is a rather awkward condition to state concisely. Firstly, we say that $f(\alpha)$ is undefined if α is $a \in N_\tau$ and $\lambda_\tau(a)$ is undefined or α is a $c!$ or $c?$ action and $\lambda(c)$ is undefined. In the case where α is a $c?x$ action and $f(\alpha)$ is undefined we must ask that the bound variable x does not appear in either (and hence both) of the source or target nodes of the transition. If the source and target nodes of a $c?x$ transition do both have x as a free variable then we notice that the use of x in the source differs from that in the target. Were a morphism f to collapse these nodes then these two uses would be, incorrectly, identified.

Suppose $n \overset{b,\alpha}{\longmapsto} n'$. We require that either $f(\alpha)$ is undefined and $\sigma(n) = \sigma(n')$ or $f(\alpha)$ is defined and $\sigma(n) \overset{b',\beta}{\longmapsto} \sigma(n')$ where $b \models_d b'$ and

$$\beta = \begin{cases} \lambda_\tau(a) & \text{if } \alpha \equiv a \\ \lambda(c)!d(e) & \text{if } \alpha \equiv c!e \\ \lambda(c)?d(x) & \text{if } \alpha \equiv c?x. \end{cases}$$

Composition of symbolic graph morphisms is given by pointwise composition of the constituent morphisms. It is a simple enough matter to check that this composition is well-defined by noting the functoriality of $\mathcal{P}$ and transitivity of $\models_d$. So we can define the category **SG** with objects as symbolic graphs and morphisms described above.

Proposition 2.3.1 *SG is a category.*

2.3.2 Symbolic graphs to transition systems

It is the intention that nodes of symbolic graphs are an abstract representation of the open terms of a value-passing language. In a similar vein, the closed terms of this language could be modelled by nodes in a transition system. It is well understood that we can close a term by substituting instantiations of values for each of its free variables and also that such an instantiation can be presented as a data environment. Therefore we ought to be able to describe a functor from **SG** to **TS** which *closes* nodes of symbolic graphs with data environments to obtain transition systems.

We call this functor the ***concretion*** functor *Conc*. Let $\mathcal{G} = (D, N, Ch, N_\tau, \longmapsto)$ be a symbolic graph. The action of *Conc* on objects is to pair up nodes of $\mathcal{G}$ with data environments and allow a transition from this pair if a symbolic edge leaving the node exists and the environment satisfies the boolean guard of this symbolic transition. Formally, $Conc(\mathcal{G}) = (N\delta, \mathcal{L}, \longrightarrow)$ where

$$N\delta = \{[n,\delta]_\asymp \mid n \in N, \delta : Var \to Val\}.$$

The equivalence relation $\asymp$ is defined so that $(n,\delta) \asymp (n,\delta')$ if and only if $\delta \restriction fv(n) = \delta' \restriction fv(n)$, and

$$\mathcal{L} = N_\tau \cup \{c!v, c?v \mid c \in Ch, v \in Val\}.$$

The transition relation $\longrightarrow$ is defined by the rules in Figure 2.1.

Conc can be lifted to morphisms very simply: $Conc(\sigma, \lambda, \lambda_\tau, d) = (\sigma_c, \lambda_c)$ where

$$\sigma_c[n,\delta] = [\sigma(n), \delta_d]$$

$$\frac{n \overset{b,\tau}{\longmapsto} n'}{[n,\delta] \xrightarrow{\tau} [n',\delta]} \qquad \delta \models b$$

$$\frac{n \overset{b,c!e}{\longmapsto} n'}{[n,\delta] \xrightarrow{c!v} [n',\delta]} \qquad \delta \models b, v = [\![e]\!]\delta$$

$$\frac{n \overset{b,c?x}{\longmapsto} n'}{[n,\delta] \xrightarrow{c?v} [n',\delta[v/x]]} \qquad \delta \models b, \text{ any } v$$

Figure 2.1. Defining the concretion functor

and (where defined)

$$\begin{aligned}
\lambda_c(a) &= \lambda_\tau(a)\\
\lambda_c(c!v) &= \lambda(c)!d(v)\\
\lambda_c(c?v) &= \lambda(c)?d(v).
\end{aligned}$$

Proposition 2.3.2 *Conc* : **SG** $\to$ **TS** *is a functor.*

Proof We first check that (σ_c, λ_c) is a well-defined morphism of transition graphs. Observe that $(n,\delta) \asymp (n,\delta')$ implies that $(\sigma(n),\delta_d) \asymp (\sigma(n),\delta'_d)$. We need to check that transitions are preserved. Suppose that $[n,\delta] \xrightarrow{\alpha} [n',\delta']$. If λ or λ_τ is undefined at α then λ_c is, moreover $\sigma(n) = \sigma(n')$. Now δ can differ from δ' by at most a bound variable in the case when α is $c?x$. But λ is undefined at α and σ identifies n and n' and, being a symbolic graph morphism, in this case we have assumed that $x \notin fv(n,n')$, hence $\sigma_c[n,\delta] = \sigma_c[n',\delta']$. Otherwise we have that λ and λ_τ are defined at α and simple checking shows that the required transition exists. For example, if α is $c!v$ then we know that $n \overset{b,c!e}{\longmapsto} n'$ for some boolean b such that $\delta \models b$ and some expression e such that $v = [\![e]\!]\delta$. Thus $\sigma(n) \overset{b',\lambda(c)!d(e)}{\longmapsto} \sigma(n')$ with $b \models_d b'$. Because $\delta \models b$ we know that $\delta_d \models b'$ so $[\sigma(n),\delta_d] \xrightarrow{\lambda(c)![\![de]\!]\delta_d} [\sigma(n'),\delta_d]$ where $[\![de]\!]\delta_d$ is just $d(v)$.

Functoriality is simple enough, clearly the identity morphisms in **SG** are mapped to the identities in **TS**. For composition we see that $Conc(f \circ g) = Conc(\sigma_f \circ \sigma_g, \ldots, d_f \circ d_g)$ where σ_f, d_f denote the obvious components of the map f. Now $Conc(f \circ g)[n,\delta] = [\sigma_f \circ \sigma_g(n), \delta_{d_f \circ d_g}]$. It is easy to see that $\delta_{d_f \circ d_g} = (\delta_{d_g})_{d_f}$ and from this it follows that $Conc(f \circ g)_1 = Conc(f)_1 \circ Conc(g)_1$. We see that $Conc(f \circ g)_2 = Conc(f)_2 \circ Conc(g)_2$ follows easily. ■

2.3.3 Symbolic graphs over 1

We now consider the specific subcategory of **SG** for which all the objects have the one-point data domain 1, all of the nodes of each graph have no free variables and all morphisms have the identity data morphism. Let us denote this category $\mathbf{SG}_1$. This subcategory is of interest because if graphs are built over a data domain with only one value this should be equivalent in some way to forgetting about values completely. We cannot expect to obtain any strong notion of equivalence between the categories $\mathbf{SG}_1$ and **TS**, because the labels on the transitions in $\mathbf{SG}_1$ are typed so that each action is a ! action or a ? action or a neutral action. The morphisms of symbolic graphs must preserve these types whereas **TS** has no such structure on its labels. Instead we look for an adjunction.

We have already described a functor *Conc* which will take us from **SG** to **TS**. Roughly, the effect this functor has on the subcategory $\mathbf{SG}_1$ is to map a symbolic graph to a transition graph with the same nodes with transitions $\overset{\mathbf{tt},c!*}{\longmapsto}$ and $\overset{\mathbf{tt},c?x}{\longmapsto}$ converted to transitions labelled $c!*$ and $c?*$. This translation into transition systems is too fine for our present purpose. We need to forget about the typing information ! and ? and, while we are in the business of forgetting, we will also forget the redundant value $*$. Define the functor $F : \mathbf{SG}_1 \to \mathbf{TS}$ as follows: On objects,

$$F(1,N,Ch,N_\tau,\longmapsto) = (N,Ch+N_\tau,\longrightarrow)$$

where

- $n \xrightarrow{a} n'$ if $n \overset{b,a}{\longmapsto} n'$ with $a \in N_\tau$ and $[\![b]\!] = \mathbf{tt}$
- $n \xrightarrow{c} n'$ if $n \overset{b,c!*}{\longmapsto} n'$ with $c \in Ch$ and $[\![b]\!] = \mathbf{tt}$
- $n \xrightarrow{c} n'$ if $n \overset{b,c?x}{\longmapsto} n'$ with $c \in Ch$ and $[\![b]\!] = \mathbf{tt}$

and on morphisms

$$F(\sigma,\lambda,\lambda_\tau,id) = (\sigma,\lambda+\lambda_\tau).$$

We look for a right adjoint to this functor. We know that any type information about a transition which has been thrown away by F must be replaced by the adjoint functor. The only guaranteed way of doing this is to map each transition $\xrightarrow{a}$ of the transition system to three transitions in the symbolic graph, one for each type. Formally then,

$$G(N,\mathcal{L},\longrightarrow) = (1,N,\mathcal{L},\mathcal{L},\longmapsto)$$

where $n \overset{\mathbf{tt},a}{\longmapsto} n'$ and $n \overset{\mathbf{tt},a!*}{\longmapsto} n'$ and $n \overset{\mathbf{tt},a?x}{\longmapsto} n'$ if $n \xrightarrow{a} n'$. We also have three corresponding symbolic transitions guarded with $\mathbf{ff}$ if $n \overset{a}{\not\longrightarrow} n'$. G acts on morphisms in the obvious way taking the map (σ,λ) to the symbolic morphism $(\sigma,\lambda,\lambda,id)$.

Proposition 2.3.3 *G is right adjoint to F.*

Proof We need to demonstrate a natural isomorphism $\phi : \mathbf{TS}(F\_,\_) \cong \mathbf{SG}_1(\_,G\_)$. Given a symbolic graph g and transition system t and a morphism $(\sigma,\lambda) : Fg \to t$ recall that the nodes of g and t are preserved by both the functors F and G so we can let ϕ preserve the node component σ. Also the label set of Fg is the disjoint sum of Ch and N_τ of g which lets us use the injections ι_1,ι_2 to pick out the apposite maps on Ch and N_τ, that is, ϕ maps (σ,λ) to $(\sigma,\lambda\circ\iota_1,\lambda\circ\iota_2,id)$. There is an obvious inverse to ϕ, namely $\phi^{-1}(\sigma,Ch,N_\tau,id) = (\sigma,[Ch,N_\tau])$. Naturality is easy to verify. ■

2.4 Value-passing CCS

We have stated that symbolic graphs can be used for modelling many value-passing calculi. We give, by way of an example, the method proposed by Hennessy and Lin [40] for constructing a symbolic graph for a process written in such a calculus. Value-passing CCS is chosen as the archetypal language to demonstrate this procedure. In order to describe the language we must first choose a data domain over which the language is defined, call this $D = (Var,\Sigma_V,I,Pr)$ and a label set for channel names, call this Ch. The abstract syntax for the language we consider is given by the grammar

$$\begin{array}{ll} t & ::= \mathbf{nil} \mid \alpha.t \mid b \to t \mid t+t \mid t|t \mid t\backslash c \mid A(\bar{e}) \\ A & ::= X \mid \lambda\bar{x}.t \end{array}$$

which contains the usual CCS operators (save for renaming) together with a boolean testing construct, $b \to$ where $b \in BoolExp$. For each constant, X, we assume a declaration of the form

$$X \Longleftarrow \lambda\bar{x}.t$$

where the free variables of t all occur in $\bar{x}$. The term $\lambda\bar{x}.t$ will be referred to as an ***abstraction***; abstractions are closed terms. Application of an abstraction f, or constant X, to a vector of data expressions $\bar{e}$ is written as $f(\bar{e})$ and $X(\bar{e})$ respectively. In these cases it will be assumed that the arity of f and X matches the length of the vector $\bar{e}$. The prefixes α are of the form τ, $c!e$ or $c?x$

$$\frac{}{\tau.p \xrightarrow{\tau} p} \qquad \frac{}{c!e.p \xrightarrow{c![\![e]\!]} p} \qquad \frac{\forall v \in \mathit{Val}}{c?x.t \xrightarrow{c?v} t[v/x]}$$

$$\frac{p \xrightarrow{\alpha} p'}{b \rightarrow p \xrightarrow{\alpha} p'} \quad \text{if } [\![b]\!] = \mathbf{tt}$$

$$\frac{p \xrightarrow{\alpha} p'}{p+q \xrightarrow{\alpha} p'} \qquad \frac{q \xrightarrow{\alpha} q'}{p+q \xrightarrow{\alpha} q'}$$

$$\frac{p \xrightarrow{\alpha} p'}{p\backslash c \xrightarrow{\alpha} p'\backslash c} \quad \text{if } \alpha \text{ doesn't use } c$$

$$\frac{p \xrightarrow{\alpha} p'}{p|q \xrightarrow{\alpha} p'|q} \qquad \frac{q \xrightarrow{\alpha} q'}{p|q \xrightarrow{\alpha} p|q'}$$

$$\frac{p \xrightarrow{c!v} p' \quad q \xrightarrow{c?v} q'}{p|q \xrightarrow{\tau} p'|q'} \qquad \frac{p \xrightarrow{c?v} p' \quad q \xrightarrow{c!v} q'}{p|q \xrightarrow{\tau} p'|q'}$$

$$\frac{t[\bar{e}/\bar{x}] \xrightarrow{\alpha} p'}{(\lambda\bar{x}.t)(\bar{e}) \xrightarrow{\alpha} p'} \qquad \frac{f(\bar{e}) \xrightarrow{\alpha} p'}{X(\bar{e}) \xrightarrow{\alpha} p'} \quad \text{if } X \Longleftarrow f$$

Figure 2.2. (Early) Operational semantics for value-passing CCS.

where $e \in \mathit{ValExp}$, $c \in \mathit{Ch}$ and $x \in \mathit{Var}$. The prefix $c?x.t$ binds occurrences of the variable x in the subterm t and we let $fv(t)$ be the set of variables of t which are not bound.

Before showing how to create a symbolic graph for this language we first present the traditional labelled transition system model. The nodes of the transition system will be *closed* terms (but not abstractions) of the language, that is terms t such that $fv(t) = \emptyset$. We will typically use letters, $p, q \ldots$ to denote closed process terms, $f, g \ldots$ to denote abstractions and $t, u, \ldots$ to denote arbitrary terms. The label set contains τ, $c!v$ and $c?v$ for all $c \in \mathit{Ch}$ and $v \in \mathit{Val}$. The transition relation is defined as the least relation satisfying the rules in Figure 2.2. We identify a process, or closed term, p with the corresponding node of the transition system described above.

The symbolic graph derived from value-passing CCS will have data domain D and arbitrary terms t will serve as nodes with $fv(t)$ being the set of variables occurring unbound in t. The label set Ch will be the channel names and there is but a single neutral action τ. The transition relation is given by the ***symbolic operational semantics*** of CCS given in Figure 2.3 Again we identify a term t with the corresponding node of the graph described above. We also make use of a function called *new* which, given a subset of *Var*, will return a new variable not present in that subset.

The transition system of value-passing CCS obtained by first compiling a symbolic graph and then applying the concretion functor of Section 2.3.2 is strictly finer than the transition system obtained directly from the operational semantics of Figure 2.2 although we will see in the next section that the two constructions are equivalent up to strong bisimulation.

2.5 Bisimulation

Bisimulation equivalence has enjoyed much success as a semantic equivalence in process theory. It is deemed to be the finest equivalence relation one need consider for the category of transition

$$\frac{}{\alpha.t \overset{\mathbf{tt},\alpha}{\longmapsto} t} \qquad \frac{t \overset{b',\alpha}{\longmapsto} t'}{(b \to t) \overset{b\wedge b',\alpha}{\longmapsto} t'}$$

$$\frac{t \overset{b,\alpha}{\longmapsto} t'}{t+u \overset{b,\alpha}{\longmapsto} t'} \qquad \frac{u \overset{b,\alpha}{\longmapsto} u'}{t+u \overset{b,\alpha}{\longmapsto} u'}$$

$$\frac{t \overset{b,\alpha}{\longmapsto} t'}{t\backslash c \overset{b,\alpha}{\longmapsto} t'\backslash c} \qquad \text{if } \alpha \text{ doesn't use } c$$

$$\frac{t \overset{b,\alpha}{\longmapsto} t'}{t|u \overset{b,\alpha}{\longmapsto} t'|u} \qquad \frac{u \overset{b,\alpha}{\longmapsto} u'}{t|u \overset{b,\alpha}{\longmapsto} t|u'} \qquad \alpha \text{ is not a } c? \text{ action}$$

$$\frac{t \overset{b,c?x}{\longmapsto} t'}{t|u \overset{b,c?y}{\longmapsto} t'[y/x]|u} \qquad \frac{u \overset{b,c?x}{\longmapsto} u'}{t|u \overset{b,c?z}{\longmapsto} t|u'[z/x]} \qquad \begin{array}{l} y = new(fv(t'|u)), \\ z = new(fv(t|u')) \end{array}$$

$$\frac{t \overset{b,c?x}{\longmapsto} t' \quad u \overset{b',c!e}{\longmapsto} u'}{t|u \overset{b\wedge b',\tau}{\longmapsto} t'[e/x]|u'} \qquad \frac{t \overset{b,c!e}{\longmapsto} t' \quad u \overset{b',c?x}{\longmapsto} u'}{t|u \overset{b\wedge b',\tau}{\longmapsto} t'[e/x]|u'}$$

$$\frac{t[\bar{e}/\bar{x}] \overset{b,\alpha}{\longmapsto} t'}{(\lambda\bar{x}.t)(\bar{e}) \overset{b,\alpha}{\longmapsto} t'} \qquad \frac{f(\bar{e}) \overset{b,\alpha}{\longmapsto} t'}{X(\bar{e}) \overset{b,\alpha}{\longmapsto} t'} \quad \text{if } X \Longleftarrow f$$

Figure 2.3. Symbolic operational semantics of CCS.

systems and has been characterised in various abstract forms [44, 56, 100, 105]. We recall the definition of (strong) bisimulation for transition systems here and consider this relation with regards to the transition system generated by the operational semantics of Figure 2.2 for value-passing CCS.

Given two transition systems $T_1 = (N_1, \mathcal{L}_1, \longrightarrow_1)$ and $T_2 = (N_2, \mathcal{L}_2, \longrightarrow_2)$ we call a relation $\mathcal{R} \subseteq N_1 \times N_2$ a ***bisimulation*** if for each $(p,q) \in \mathcal{R}$ we have

- whenever $p \xrightarrow{\alpha}_1 p'$ then $q \xrightarrow{\alpha}_2 q'$ for some q' such that $(p',q') \in \mathcal{R}$
- whenever $q \xrightarrow{\alpha}_2 q'$ then $p \xrightarrow{\alpha}_1 p'$ for some p' such that $(p',q') \in \mathcal{R}$

We say that p and q are bisimulation equivalent or ***bisimilar***, written $p \sim q$, if there exists a bisimulation containing (p,q).

In the previous section we showed how to construct a transition system from the value-passing language CCS. If we use bisimulation equivalence on this transition system then we obtain a notion of equivalence for process terms, that is p and q as processes are equivalent iff $p \sim q$ as nodes of the transition system for value-passing CCS. This notion of equivalence has been dubbed ***early bisimulation*** in the literature [75]. Bisimulation equivalence can be lifted to abstractions by defining $f \sim g$ iff, for all values $\bar{v}$, we have $f(\bar{v}) \sim g(\bar{v})$.

As promised we now show that the transition systems constructed directly and by concreting a symbolic graph are equivalent up to bisimulation. In the following, $t\delta$ will represent the closed process obtained from t by substituting its free variables by the corresponding values of δ (and thus will represent the node of the transition system for that closed process). We will also use the notation $[t,\delta]$ of Section 2.3.2 and $\equiv$ to denote syntactic identity.

Proposition 2.5.1

(1) $t\delta \xrightarrow{\alpha} p'$ *implies* $[t,\delta] \xrightarrow{\alpha} [t',\delta']$ *for some* t',δ' *such that* $p' \equiv t'\delta'$.

(2) $[t,\delta] \xrightarrow{\alpha} [t',\delta']$ *implies* $t\delta \xrightarrow{\alpha} t'\delta'$.

Proof Easy induction on the derivation of the transitions. ■

Corollary 2.5.2 $p \sim [t,\delta]$ *whenever* $p \equiv t\delta$.

There seem to be two acceptable generalisations of bisimulation equivalence when talking about value-passing languages; one of which is early bisimulation equivalence presented above, its counterpart being late bisimulation [75]. Late bisimulation is strictly finer than early bisimulation so we give a typical example of how they differ. Consider the processes over $\mathcal{N}$

$$p \Longleftarrow (c?x.c!x.\mathbf{nil}) + (c?x.(x \geq 0 \rightarrow \mathbf{nil}))$$

and

$$q \Longleftarrow (c?x.(x > 0) \rightarrow c!x.\mathbf{nil}) + (c?x.(x = 0) \rightarrow c!x.\mathbf{nil}).$$

It is quite easy to see that p and q are early bisimilar because both p and q have the ability to receive a value and then output it again on channel c or to receive a value and then deadlock. Bisimulation is designed to be a branching time equivalence, meaning that it respects the branching structure of processes, but closer inspection of the above example reveals that there is some subtlety involved in interpreting *respects the branching structure* when dealing with value-based languages. To match the move $p \xrightarrow{c?v} c!v.\mathbf{nil}$ we choose a move from q but this choice is contingent upon the value v — if v is 0 then we can take the right branch otherwise we can take the left branch. We see that, when using an early semantics, the branch of q used to match the input actions from the left branch of p may vary with the value being received. One might consider this property

$$\frac{}{\tau.p \xrightarrow{\tau} p} \qquad \frac{}{c!e.p \xrightarrow{c![\![e]\!]} p} \qquad \frac{}{c?x.t \xrightarrow{c?} (x)t}$$

$$\frac{p \xrightarrow{\alpha} p'}{b \rightarrow p \xrightarrow{\alpha} p'} \quad \text{if } [\![b]\!] = \mathbf{tt}$$

$$\frac{p \xrightarrow{\alpha} p'}{p+q \xrightarrow{\alpha} p'} \qquad \frac{q \xrightarrow{\alpha} q'}{p+q \xrightarrow{\alpha} q'}$$

$$\frac{p \xrightarrow{\alpha} p'}{p\backslash c \xrightarrow{\alpha} p'\backslash c} \quad \text{if } \alpha \text{ doesn't use } c$$

$$\frac{p \xrightarrow{\alpha} p'}{p|q \xrightarrow{\alpha} p'|q} \qquad \frac{q \xrightarrow{\alpha} q'}{p|q \xrightarrow{\alpha} p|q'} \qquad \alpha \text{ is not a } c? \text{ action}$$

$$\frac{p \xrightarrow{c?} (x)t}{p|q \xrightarrow{c?} (x)(t|q)} \qquad \frac{q \xrightarrow{c?} (x)u}{p|q \xrightarrow{c?} (x)(p|u)}$$

$$\frac{p \xrightarrow{c?} (x)t \quad q \xrightarrow{c!v} q'}{p|q \xrightarrow{\tau} t[v/x]|q'} \qquad \frac{p \xrightarrow{c!v} p' \quad q \xrightarrow{c?} (x)u}{p|q \xrightarrow{\tau} p'|u[v/x]}$$

$$\frac{t[\bar{e}/\bar{x}] \xrightarrow{\alpha} p'}{(\lambda\bar{x}.t)(\bar{e}) \xrightarrow{\alpha} p'} \qquad \frac{f(\bar{e}) \xrightarrow{\alpha} p'}{X(\bar{e}) \xrightarrow{\alpha} p'} \text{ if } X \Longleftarrow f$$

Figure 2.4. (Late) Operational semantics for value-passing CCS.

undesirable of an equivalence which is meant to respect branching structure. This property holds purely because the early semantics of Figure 2.2 actually treat the term $c?x.t$ as if it were a sum

$$\sum_{v \in Val} c?v.t[v/x]$$

thereby confusing the branching structure of the original process term. To see how this summation confuses we need only look at the transition systems for p and q: they are not only bisimilar but actually isomorphic!

Late bisimulation is a remedy to this problem and as such clearly requires a different operational semantics, we give these in Figure 2.4. Notice that the action $c?v$ has now been decomposed into the action $c?$ and substitution (*cf.* the communication rule). The term $(x)t$ has also been called an abstraction [75] and may be thought of as a function from *Val* to closed terms; to avoid confusion with the previous use of *abstraction* we will, wherever necessary, refer to $(x)t$ as an ***?-abstraction***. The result of a communication is to pass a value across a parallel, apply this function to it and β reduce. We can now define the notion of late bisimulation using the late operational semantics. We build a transition system similar to the one of Section 2.4 by taking the nodes to be the closed process terms of the language along with ?-abstractions $(x)t$. The transition relation $\longrightarrow_L$ is defined so that $p \xrightarrow{\alpha}_L r$ (where r is a closed term or ?-abstraction) according to the rules in Figure 2.4 and for each v we have $(x)t \xrightarrow{@v}_L t[v/x]$, where $@v$ is a new kind of action. The relation $\sim$ on this transition system is called ***late bisimulation*** and can be characterised more simply on closed processes in the following way.

Processes p_0 and q_0 are late bisimilar if there exists a relation $\mathcal{R}$ containing (p_0, q_0) such that for each $(p,q) \in \mathcal{R}$ (omitting symmetric rules for q)

- whenever $p \xrightarrow{\alpha} p'$ (for α not $c?$) then $q \xrightarrow{\alpha} q'$ for some q' such that $(p', q') \in \mathcal{R}$
- whenever $p \xrightarrow{c?} (x)t$ then $q \xrightarrow{c?} (y)u$ for some $(y)u$ such that $(t[v/x], u[v/y]) \in \mathcal{R}$ for all $v \in Val$

This is in fact the standard definition of late bisimulation in the literature. We will denote both early and late bisimilarity as $p \sim q$ or as $p \sim_E q$ and $p \sim_L q$ respectively, whenever the use is unclear from the context. There is of course an analogous result to Proposition 2.5.1 for the late semantics and a modified late version of the concretion functor. But we forgo the details here. Returning to our example though we see that p and q fail to be late bisimilar because any late bisimulation containing them must be able to match $p \xrightarrow{c?} (x)(c!x.\mathbf{nil})$. Clearly there are only two possible matches $q \xrightarrow{c?} (x)(x > 0 \rightarrow c!x.\mathbf{nil})$ and $q \xrightarrow{c?} (x)(x = 0 \rightarrow c!x.\mathbf{nil})$ neither of which suffice.

2.5.1 Symbolic bisimulation

So far we have shown how to construct a transition system from CCS and have given an equivalence called bisimulation on this transition system. We have also shown how to construct a symbolic graph from CCS and then *concrete* it to a transition system bisimilar to the directly constructed system. What is missing from this picture is an equivalence at the symbolic level. That is, a definition of bisimulation for symbolic graphs which is preserved by the concretion functor — this is symbolic bisimulation.

Unfortunately symbolic graphs do not come equipped with substitution and α-conversion. This causes a slight problem for giving a definition of bisimulation for them. For example, consider the following two graphs

$$t \overset{c?x}{\longmapsto} t' \overset{c!x}{\longmapsto} \cdot$$

and

$$u \overset{c?y}{\longmapsto} u' \overset{c!y}{\longmapsto} \cdot$$

Now any reasonable notion of bisimulation ought to identify t and u as they differ only by a bound variable. One might expect that to match $t \overset{c?x}{\longmapsto} t'$ we use $u \overset{c?y}{\longmapsto} u'$ and then ask that $t'[z/x]$ and $u'[z/y]$ be related for some fresh variable z. The problem is how to interpret $t'[z/x]$.

Hennessy and Lin define what they call a *term* to be a node of a symbolic graph, paired up with a substitution $\sigma : Var \rightarrow Var$. They proceed to define symbolic bisimulations on terms rather than on the nodes of graphs themselves. The approach we take is to identify a class of graphs for which such an expression would make sense, that is, those graphs which for each node n and substitution $[z/x]$ there is a node n' which behaves like n under this substitution. This property is easily defined using the construction described below which saturates a graph with substitutions.

Given a graph $G = (D, N, Ch, N_\tau, \longmapsto)$ we create a graph $SSat(G) = (D, N^{ss}, Ch, N_\tau, \longmapsto_{ss})$ where

$$N^{ss} = \{(n, \sigma) \mid n \in N, \sigma : Var \rightarrow Var\}$$

and $\longmapsto_{ss}$ is defined by the following rules.

$$n \overset{b,\tau}{\longmapsto} n' \text{ implies } (n,\sigma) \overset{b\sigma,\tau}{\longmapsto}_{ss} (n', \sigma)$$
$$n \overset{b,c!e}{\longmapsto} n' \text{ implies } (n,\sigma) \overset{b\sigma,c!e\sigma}{\longmapsto}_{ss} (n', \sigma)$$
$$n \overset{b,c?x}{\longmapsto} n' \text{ implies } (n,\sigma) \overset{b\sigma,c?z}{\longmapsto}_{ss} (n', \sigma[z/x]) \text{ where } z = new(\{\sigma(y) \mid y \in fv(n)\})$$

We may consider N^{ss} as a transition system, with the pairs b, α as labels, and therefore have a bisimulation equivalence relation $\sim_{ss}$ on N^{ss}. Let $SSat(G)/\sim_{ss}$ be the graph $SSat(G)$ quotiented by this equivalence relation.

We say that a graph G is ***substitution saturated*** if G is isomorphic to $SSat(G)/\sim_{ss}$. Here isomorphism means there are mutually inverse graph morphisms whose components for channel renaming and data are identities. Note that for each node n of a substitution saturated graph, G, and for each $\sigma : Var \rightarrow Var$ there is a node (n, σ) in $SSat(G)$ and, via the isomorphism, a node n' in G which behaves exactly like $n\sigma$. This fact enables us to write $n\sigma$ unambiguously for substitution saturated graphs.

Proposition 2.5.3 *(1) The symbolic graph for CCS is substitution saturated.*

(2) $SSat(G)/\sim_{ss}$ is substitution saturated for all G.

Proof (1) A term t is mapped to the equivalence class $[t, id]$ and, inversely, the class $[t, \sigma]$ is mapped to the term $t\sigma$. This is an isomorphism because for any t', σ such that $t \equiv t'\sigma$ we know that $(t, id) \sim_{ss} (t', \sigma)$. This can be shown by constructing a relation

$$\mathcal{R} = \{(t, \sigma), (t', \sigma' \circ \sigma) \mid t\sigma \equiv t'\sigma' \circ \sigma\}$$

and using structural induction on t to show that $R \subset \sim_{ss}$.

(2) We need to show that $SSat(G)/\sim_{ss}$ is isomorphic to $SSat[SSat(G)/\sim_{ss}]/\sim_{ss}$. To embed the former in the latter we simply take $[n, \sigma]$ to $[[n, \sigma], id]$. This has an inverse which maps $[[n, \sigma], \xi]$ to $[n, \xi \circ \sigma]$. Of course this uses the fact that $[[n, \sigma], id]$ is bisimilar to $[[n, \sigma'], \xi]$ whenever $\sigma = \xi \circ \sigma'$. ■

So although we cannot define symbolic bisimulation on graphs in general we can at least use the previous proposition to define symbolic bisimulation upon the saturation of an arbitrary graph. Given two substitution saturated graphs $(D, N_1, Ch_1, N_{\tau_1}, \longmapsto_1)$ and $(D, N_2, Ch_2, N_{\tau_2}, \longmapsto_2)$ over the same data domain D, let $\mathbf{S} = \{S^b \subseteq N_1 \times N_2\}$ be a boolean indexed family of relations. The booleans here do not necessarily belong to $BoolExp$ but rather the boolean metalanguage used to reason about data properties. Define $\mathcal{SB}(\mathbf{S})$ to be the family of relations such that

$(t, u) \in \mathcal{SB}(\mathbf{S})^b$ if whenever $t \overset{b_1,\alpha}{\longmapsto}_1 t'$ there exists a variable z such that $z \notin fv(b, t, u)$ and a finite collection of booleans B such that $b \wedge b_1 \models \bigvee B$, so B is a $b \wedge b_1$*-partition*, and for each $b' \in B$ there exists a $u \overset{b_2,\beta}{\longmapsto}_2 u'$ such that $b' \models b_2$ and

- if $\alpha \in N_\tau$ then $\beta \equiv \alpha$ and $(t', u') \in S^{b'}$
- if α is $c!e$ then $\beta \equiv c!e'$ with $b' \models e = e'$ and $(t', u') \in S^{b'}$
- if α is $c?x$ then $\beta \equiv c?y$ for some y and $(t'[z/x], u'[z/y]) \in S^{b'}$

There is of course a symmetric condition on transitions from u.

A family of relations $\mathbf{S}$ is called an ***early symbolic bisimulation*** if $\mathbf{S} \subseteq \mathcal{SB}(\mathbf{S})$. Furthermore, if the fresh variable z never appears in the partition B then the family $\mathbf{S}$ is a ***late symbolic bisimulation***.

We write $t \sim_E^b u$ (respectively $t \sim_L^b u$) if there is an early (late) symbolic bisimulation $\mathbf{S}$ such that $(t, u) \in S^b$.

We now give an example of two early bisimilar processes, p and q. We demonstrate that they are related in the most general boolean world tt.

$$\begin{tikzcd}
 & \bullet & \\
p \arrow[ur, "c?x"] \arrow[dr, "c?x"'] & & \\
 & p' \arrow[r, "c!x^2"] & \bullet
\end{tikzcd}
\qquad
\begin{tikzcd}
 & q' \arrow[dr, "odd(x)\text{, } c!x^2"] & \\
q \arrow[ur, "c?x"] \arrow[dr, "c?x"'] & & \bullet \\
 & q'' \arrow[ur, "even(x)\text{, } c!4(x/2)^2"'] &
\end{tikzcd}$$

To show $p \sim^{\mathbf{tt}} q$ we must find matching partitions and moves for each of the transitions from both p and q. We describe the matches for the transition $p \overset{\mathbf{tt},c?x}{\longmapsto} p'$. What we require is a tt-partition such that for each boolean b in this partition we know that b guarantees a $c?x$ transition from q to a node which is at least b related to p'. The partition we require then is $\{even(x), odd(x)\}$. If we take $even(x)$ we see that $q \overset{\mathbf{tt},c?x}{\longmapsto} q''$ with $p' \sim^{even(x)} q''$ and for $odd(x)$ we use $q \overset{\mathbf{tt},c?x}{\longmapsto} q'$ with $p' \sim^{odd(x)} q'$. No further partitioning is required and we note that the bound variable x occurs in the partition making this an early rather than late bisimulation.

We previously suggested that symbolic bisimulation ought to be preserved by concretion. This amounts to saying that

$$t \sim^{\mathbf{tt}} u \text{ implies } t\delta \sim u\delta \text{ for all } \delta.$$

The relationship between concrete and symbolic bisimulation is in fact much tighter.

Proposition 2.5.4 *$t \sim^b u$ if and only if $t\delta \sim u\delta$ for all $\delta \models b$.*

Proof See [40] ■

2.6 Symbolic graphs with assignment

A shortcoming of symbolic graphs is that, although they are good at modelling infinitely branching processes finitely, there are still many value-passing processes which are intuitively finite in structure, but are modelled by infinite symbolic graphs. For example, the process $X(0)$ declared by

$$X \Longleftarrow \lambda x.c!x.X(x+2)$$

repeatedly outputs the sequence of even numbers on c. The structure of this process is very simple: there is a single *state* from which there is a c output transition. The actual transition graph, which also happens to be the symbolic graph, for this process looks like

$$X(0) \xrightarrow{c!0} X(2) \xrightarrow{c!2} X(4) \xrightarrow{c!4} \cdots$$

An infinite graph is being used to model a relatively simple structure.

One approach to rectifying this situation was presented, independently, by Lin and Paczowski, [65, 78]. There was also a similar solution prescribed in [96] using a different formalism to symbolic graphs. Lin and Paczowski's solution involved introducing explicit assignments, or substitutions into the arcs of the graphs. Thus regular behaviours such as the example $X(0)$ can be described by their transitions along with a substitution to describe how the data part of the process is affected by transition. The symbolic graph with assignment for $X(0)$ now looks like

$$\begin{tikzcd}
\bullet \arrow[r, "{x:=0\,,\,c!x}"] & \{x\} \arrow[loop right, "{x:=x+2\,,\,c!x}"]
\end{tikzcd}$$

More generally, a symbolic graph with assignment is a symbolic graph whose edges are now labelled with a triple (b, θ, α) where $b \in BoolExp$, $\alpha \in Act$ and θ is an assignment $\bar{x} := \bar{e}$. We write $m \overset{b,\theta,\alpha}{\longmapsto} n$ to denote arcs of the graph and ask that $fv(b, \bar{e}) \subseteq fv(m)$, $fv(\alpha) \subseteq \bar{x}$ and $fv(n) \subseteq \bar{x} \cup bv(\alpha)$.

Symbolic graphs with assignment can be unfolded into symbolic graphs in a similar manner to saturating a symbolic graph with substitutions. Rather than using simple substitutions, however, we saturate with arbitrary substitutions. Thus, we can create a symbolic graph, roughly, by the rule

$$\frac{m \overset{b,\theta,\alpha}{\longmapsto} n}{(m,\sigma) \overset{b\sigma,\alpha\theta\sigma}{\longmapsto} (n, \theta\sigma)}$$

This allows us to define symbolic bisimulation over graphs with assignment simply by asking that their respective symbolic graphs be bisimilar.

An important feature of graphs with assignments is that any term of regular, value-passing CCS, that is, the sublanguage without parallel composition and restriction, is modelled by a *finite* graph with assignment. This fact is exploited in [65, 78] in order to present an algorithm for reducing the decidability of strong bisimulation down to deciding validity of boolean expressions about data. The particular language that these boolean expressions are described in is essentially first-order predicate logic with parameterised fixpoints.

In Chapter 6 we present a proof system for reasoning about regular value-passing CCS processes. We don't use symbolic graphs with assignment explicitly although one might consider the *standard declarations* of that chapter as a syntax for such graphs. However, we do make use of the algorithms of [65, 78] for calculating boolean expressions which guarantee bisimilarity between finite graphs with assignment. Moreover, we also make use of the fact that the class of finite symbolic graphs with assignment is closed under parallel composition, whereas this is certainly not the case for finite symbolic graphs. We demonstrate this with the following example: consider the graphs

$$\begin{tikzcd}
\bullet \arrow[d, "c!0"'] & \bullet \arrow[d, "c?x"'] & \bullet \arrow[d, "d?y"'] \\
\bullet & \bullet \arrow[d, "o!x"'] & \bullet \arrow[u, bend right, "c!y"'] \\
 & \bullet \arrow[uu, bend right, "d!(x+1)"'] &
\end{tikzcd}$$

Their parallel composition, hiding channels c and d, as defined in [65], expands to the infinite symbolic graph

$$\bullet \xrightarrow{\tau} \bullet \xrightarrow{o!0} \bullet \xrightarrow{\tau} \bullet \xrightarrow{\tau} \bullet \xrightarrow{o!1} \bullet \xrightarrow{\tau} \cdots$$

for a process which implements an incremental counter, outputting on channel o.

Chapter 3

Strong Bisimulation for a Calculus of Broadcasting Systems

We turn to the world of broadcasting systems for our first demonstration of the symbolic technique. The language we consider is CBS, a value-passing process calculus where communication between agents is effected by the broadcasting of values. The language is similar in style to value-passing CCS but has a multiway synchronisation operator in place of the handshaking parallel of CCS. Another departure from CCS is the lack of channel names. The channel names of CCS explicitly name the medium, or channel, on which each communication takes place. This gives rise to the notions of local channels and a communication topology. Such a description of communication sits uncomfortably in a broadcast model. The idea behind broadcasting is that communication takes place between *all* of the agents in a network and not just ones with particular communication capabilities on a given channel, *i.e.*, the communication topology is discrete. So we dispense with channel names entirely and assume that all communication takes place in some medium called the ether. Local message passing can still be achieved by tagging the messages with an identifier or translating the messages into messages which can only be understood by select processes. For example, following Prasad's analogy with human conversation, in a room full of french speaking people, if an individual were to start speaking in esperanto they would find that their message would not be received throughout the room but only by a lucky few. This approach to value-passing necessitates the use of *pattern-matching*. Pattern-matching in value-passing calculi is typically conceived as a post-reception boolean test; the recipient deciding, having received the value, what action to take. In contrast to this the pattern-matching in CBS can also be done prior to reception. A process not intended to receive a value will simply not receive it and its state will remain unchanged. In some sense the pattern-matching is being done by the ether.

The ideas behind the Broadcast Calculus are due to Prasad [85, 86, 87]. We are concerned with applying the symbolic techniques described to provide the calculus with a finitary equational theory and proof system for establishing process identities. Existing work [86] introduces a notion of both strong and weak bisimulation for CBS. Also, a complete axiomatisation for the strong equivalence is given for a pure version of the calculus. For the remainder of this chapter we reinvestigate the notion of strong equivalence for CBS and use symbolic techniques to give a complete axiomatisation of this equivalence. In order to use the symbolic approach we give a symbolic operational semantics for CBS. These semantics do not define a symbolic graph as in Section 2.3 because of the pattern-matching present in the language. However, we will see later that a slight modification of symbolic bisimulation allows the technique to be applied smoothly.

3.1 The broadcast calculus

The calculus we consider is a minor variation on that of [86]. The syntax is described by the following grammar:

$$t ::= \mathbf{O} \mid e!t \mid x \in S?t \mid b \gg t \mid \sum_{i \in I} t_i \mid t|t \mid t_{(f,g)} \mid A(\bar{e}).$$

It has many of the usual operators of *CCS*, [74] but communication is achieved by broadcasting values to all processes in the environment. The language is of course a value-passing language and as such the syntax presupposes a data domain D. Thus we will freely use the terms *Val*, *ValExp*, *etc.* The process $e!p$ broadcasts the value of the data expression e while $x \in S?t$ is a process which, on hearing the value v proceeds to act like the process $t[v/x]$ provided $v \in S$; otherwise the value is ignored.

Let τ be a special value in *Val*; τ represents *noise* in the system, i.e. broadcasts of values which can not be deciphered by any process. We ask that the sets S guarding input prefixes never contain the value τ. In general v will range over non-τ values and w will range over all of *Val*, including τ. The restriction and renaming operators of CCS are replaced by the one *translation* construct $t_{(f,g)}$. Here both f and g are functions from *Val* to *Val*, strict in the sense that $f(\tau) = g(\tau) = \tau$. In $t_{(f,g)}$ the behaviour of t is translated using f for use by the environment while the behaviour of the environment is translated by g for use by t. The strictness condition enforces the constraint that noise cannot be translated into an interpretable value.

The operational semantics for this language, *CBS* is given in Figure 3.1; it more or less coincides with that presented in [86]. We note that this is an *early* semantics and that the notion of equivalence that we derive from these will in fact be an early form of bisimulation. Throughout we assume that with each constant name A we have an associated definition: $A \Longleftarrow \lambda\bar{x}.t_A$ where $\bar{x}$ contains all of the free variables that appear in t_A, and A always occurs within the scope of an action prefix in t_A. The rules determine three different kinds of binary relations over agents, $p \xrightarrow{v?} q$ representing the effect of inputting a value v, $p \xrightarrow{w!} q$, $w \in Val$, representing the output of the value w and the novel discard relation $p \xrightarrow{w:} q$. The reader is referred to [86] for more explanation and discussion of these rules.

The most notable difference between the operational semantics of *CCS*, [74], and *CBS* is this discard relation. It is essentially a *negation* of the transition $p \xrightarrow{v?} p'$ for some p' (see Lemma 3.1.1 below) and is used to facilitate the presentation of the semantics for the parallel operator.

Some simple properties of these relations are given in the following lemma:

Lemma 3.1.1 *For every agent p*

- *if* $p \xrightarrow{w:} q$ *then* q *is* p
- $p \xrightarrow{v:} p$ *if and only if there does not exist a* q *such that* $p \xrightarrow{v?} q$
- $p \xrightarrow{\tau:} p$

Proof By induction on the rules of inference in Figure 3.1. ■

Intuitively a process discards a value when it is in a state in which values can not be received. So the first property of this Lemma is very natural: ignoring or discarding a broadcasted value does not change the state of a process. The second property states that discarding a value is exactly the same as not being able to receive it. The final property states that all processes ignore noise. This relies on the fact that in all guarded input terms $x \in S?t$, S is a subset of *Val* not containing τ.

Using the operational semantics given above we can describe a labelled transition system semantics for CBS. Such a semantics would be similar to those given for (early) value-passing CCS in Chapter 2 but with a notable difference. It is no longer true that $p \xrightarrow{v?} q$ implies that for

Discard	Input	Output
$\mathbf{0} \xrightarrow{w:} \mathbf{0}$		
$\frac{w \notin S}{x \in S?t \xrightarrow{w:} x \in S?t}$	$\frac{v \in S}{x \in S?t \xrightarrow{v?} t[v/x]}$	
$e!p \xrightarrow{w:} e!p$		$\frac{[\![e]\!] = w}{e!p \xrightarrow{w!} p}$
$\frac{\forall i \in I \cdot p_i \xrightarrow{w:} p_i}{\sum_I p_i \xrightarrow{w:} \sum_I p_i}$	$\frac{\exists i \in I \cdot p_i \xrightarrow{v?} p'}{\sum_I p_i \xrightarrow{v?} p'}$	$\frac{\exists i \in I \cdot p_i \xrightarrow{w!} p'}{\sum_I p_i \xrightarrow{w!} p'}$
$\frac{[\![b]\!] = \mathbf{ff}}{b \gg p \xrightarrow{w:} b \gg p}$		
$\frac{p \xrightarrow{w:} p}{b \gg p \xrightarrow{w:} b \gg p}$	$\frac{p \xrightarrow{v?} p' \quad [\![b]\!] = \mathbf{tt}}{b \gg p \xrightarrow{v?} p'}$	$\frac{p \xrightarrow{w!} p' \quad [\![b]\!] = \mathbf{tt}}{b \gg p \xrightarrow{w!} p'}$
$\frac{t_A[\bar{e}/\bar{x}] \xrightarrow{w:} t_A[\bar{e}/\bar{x}]}{A(\bar{e}) \xrightarrow{w:} A(\bar{e})}$	$\frac{t_A[\bar{e}/\bar{x}] \xrightarrow{v?} p'}{A(\bar{e}) \xrightarrow{v?} p'}$	$\frac{t_A[\bar{e}/\bar{x}] \xrightarrow{w!} p'}{A(\bar{e}) \xrightarrow{w!} p'}$
$\frac{p \xrightarrow{gw:} p}{p_{(f,g)} \xrightarrow{w:} p_{(f,g)}}$	$\frac{p \xrightarrow{gv?} p'}{p_{(f,g)} \xrightarrow{v?} p'_{(f,g)}}$	$\frac{p \xrightarrow{w!} p'}{p_{(f,g)} \xrightarrow{fw!} p'_{(f,g)}}$

$$\frac{p \xrightarrow{\alpha} p' \quad q \xrightarrow{\beta} q'}{p|q \xrightarrow{\alpha\bullet\beta} p'|q'} \qquad \alpha\bullet\beta \neq \perp$$

$\bullet$	$w!$	$w?$	$w:$
$w!$	$\perp$	$w!$	$w!$
$w?$	$w!$	$w?$	$w?$
$w:$	$w!$	$w?$	$w:$

Figure 3.1. Operational semantics for closed agents.

every value v' there is a process $q_{v'}$ such that $p \xrightarrow{v'?} q_{v'}$. One reason is the use of guarded inputs, $x \in S?t$ which are conspicuously absent from CCS; here a value can be input only if it is in S. However even if the only input construct allowed is $x \in Val?t$ the property still does not hold. For example the process $(x \in Val?t)_{(f,g)}$ can only receive the values from Val which g doesn't map to τ.

One might reasonably ask if this discard behaviour might be modelled in CCS using reception, boolean testing and recursion. That is, $x \in S?t$ might be written as

$$X \Longleftarrow c?x(x \in S \rightarrow t) + (x \notin S \rightarrow X).$$

This has a behaviour corresponding to $x \in S?t$ in CBS but with the crucial difference that any move from X in a choice context will resolve that choice, whereas discard moves from $x \in S?t$ will not.

3.1.1 Strong bisimulation for CBS

Given a transition system semantics for CBS there is a standard definition of bisimulation which can be applied to this transition system. This would yield a notion of equivalence, identical to Prasad's strong bisimulation, for the broadcast calculus but we question whether this equivalence is appropriate in this setting. For instance, the use of handshaking communication in CCS requires the protocol that in order for a value to be communicated the sending process must synchronise with a unique receiver. The observable effect is that the resulting action of this communication is hidden using an internal τ action. So if a process' environment offers to send it a value we can directly observe whether the process chooses to receive the value by observing whether the handshake was effected. In CBS the situation is a little different; there is no notion of an internal action so the observed result of a communication is simply a value being offered. If a CBS process' environment offers to send it a value we have no recourse to the communication protocol to observe whether the process chooses to receive it or ignore it, the observed effect is identical in both cases. With this in mind it is unclear at best whether the receipt of a value should be treated as an observable action in CBS. What we would like then is a means of defining a notion of bisimulation equivalence based on broadcast actions alone. We find this in *barbed bisimulation.*

Barbed bisimulation was proposed in [94] as a means of finding an appropriate definition of bisimulation for the higher-order π-calculus. The approach is straightforward and uncontroversial since it relies only on a notion of reduction, which we have in $\xrightarrow{\tau!}$, and a notion of when agents have the ability to produce values, which we have in $\xrightarrow{v!}$. This satisfies our criterion that the definition uses broadcast actions only.

For any value v let $p \downarrow v$ mean that $p \xrightarrow{v!} p'$ for some p'. Then a relation $\mathcal{R}$ between agents is called a ***barbed bisimulation*** if for each $(p,q) \in \mathcal{R}$ we have

- whenever $p \xrightarrow{\tau!} p'$ then $q \xrightarrow{\tau!} q'$ for some q' such that $(p',q') \in \mathcal{R}$
- whenever $q \xrightarrow{\tau!} q'$ then $p \xrightarrow{\tau!} p'$ for some p' such that $(p',q') \in \mathcal{R}$
- $p \downarrow v$ if and only if $q \downarrow v$.

We use $\sim_{barb}$ to denote the maximal such relation which is obviously an equivalence. However it is itself not very interesting as a semantic equivalence as it is an extremely coarse relation. For example, any processes $x \in S?t$ and $y \in S'?u$ will be $\sim_{barb}$ related. The equivalence we are interested in is the congruence generated by $\sim_{barb}$. That is, we relate processes which can broadcast the same values when sitting in a common context.

For agents p and q let $p \sim^c_{barb} q$ if $C[p] \sim_{barb} C[q]$ for every CBS context $C[\_]$. We will sometimes want to specify over which data language the CBS contexts are built. To indicate that a particular value set V is used we write $(p \sim^c_{barb} q) : V$

This is a fairly intractable definition of equivalence so the remainder of this section is devoted to giving a simple bisimulation type characterisation of $\sim^c_{barb}$. In order to do this we appeal to our intuitive ideas about a process reacting to a broadcast value by indistinguishably accepting the value or ignoring it, and define the ***reaction*** $v??$ as follows:

let $p \xrightarrow{v??} q$ if $p \xrightarrow{v?} q$ or $p \xrightarrow{v:} q$.

With this new arrow we define a new kind of bisimulation relation. A relation $\mathcal{R}$ between agents is called a ***noisy bisimulation*** if for each $(p,q) \in \mathcal{R}$ we have

- whenever $p \xrightarrow{w!} p'$ then $q \xrightarrow{w!} q'$ for some q' such that $(p',q') \in \mathcal{R}$
- whenever $p \xrightarrow{v??} p'$ then $q \xrightarrow{v??} q'$ for some q' such that $(p',q') \in \mathcal{R}$
- whenever $q \xrightarrow{w!} q'$ then $p \xrightarrow{w!} p'$ for some p' such that $(p',q') \in \mathcal{R}$
- whenever $q \xrightarrow{v??} q'$ then $p \xrightarrow{v??} p'$ for some p' such that $(p',q') \in \mathcal{R}$.

We let $p \sim_n q$ if there exists some noisy bisimulation $\mathcal{R}$ such that $(p,q) \in \mathcal{R}$.

Because of Lemma 3.1.1 noisy bisimulations can be simplified considerably:

Proposition 3.1.2 *Let $\mathcal{R}$ be a relation over pairs of agents. Then $\mathcal{R}$ is a noisy bisimulation if and only if for each $(p,q) \in \mathcal{R}$ we have*

- *whenever $p \xrightarrow{w!} p'$ then $q \xrightarrow{w!} q'$ for some Q' such that $(p',q') \in \mathcal{R}$*
- *whenever $p \xrightarrow{v?} p'$ then $q \xrightarrow{v??} q'$ for some q' such that $(p',q') \in \mathcal{R}$*

(Plus symmetric conditions on q).

Proof Suppose $\mathcal{R}$ satisfies the conditions of the Proposition. We need only check that a discard move $p \xrightarrow{v:} p'$, where $(p,q) \in \mathcal{R}$, can be matched by a move from q. We know from the first part of Lemma 3.1.1 that p' must be p. In fact if $q \xrightarrow{v:} q'$ then q' must also be q and we are done. Therefore we assume that $q \not\xrightarrow{v:} q$. The second part of Lemma 3.1.1 tells us that $q \xrightarrow{v?} q'$ for some q'. This implies, using the second property of $\mathcal{R}$, that $p \xrightarrow{v??} p'$ with $(p',q') \in \mathcal{R}$. It follows, again from Lemma 3.1.1, that $p \not\xrightarrow{v?}$ and so we know that p' is p and $(p,q') \in \mathcal{R}$. ■

The reader should note that this noisy bisimulation equivalence differs from the strong bisimulation equivalence proposed by Prasad in [86]. Unlike strong bisimulation it turns out that, because of the pre-emptive power of the reception of inputs, noisy bisimulation is not preserved by the choice operator. For example $x \in Val?\mathbf{O} \sim_n \mathbf{O}$ but $v!\mathbf{O} + x \in Val?\mathbf{O} \not\sim_n v!\mathbf{O} + \mathbf{O}$. However it is preserved by all of the other operators.

Proposition 3.1.3 *The relation $\sim_n$ is preserved by all of the CBS operators except choice.*

Proof As in [74], to show that noisy bisimulation is preserved by composition, say, we simply let

$$\mathcal{R} = \{(p|r),(q|r) \mid \text{for all } p,q,r \text{ such that } p \sim_n q\}$$

and show that $\mathcal{R}$ is a noisy bisimulation. The other operators are treated in a similar way. ■

We can also capture noisy bisimulation equivalence from $\sim_{barb}$ using static contexts, *i.e.* contexts in which the *hole* does not appear in sub-terms of the form $t+u$. In the following proposition we need to consider a larger value set than the underlying set Val. More specifically we use the set $Val^+ \overset{def}{=} Val \cup Val' \cup \{a,b\}$, where Val' is a set of values such that for each $v \in Val$ (save τ) there exists exactly one v' in Val' with $v' \notin Val$ and $a,b \notin Val \cup Val'$. The contexts that we use to obtain noisy bisimulation are defined over Val^+ rather than Val.

Proposition 3.1.4 *If $C[p_0] \sim_{barb} C[q_0]$ for every static context $C[\_]$ defined over Val^+ then $p_0 \sim_n q_0$.*

Proof Given p_0, q_0 defined over a value set Val, we suppose that $C[p_0] \sim_{barb} C[q_0]$ for every static context C defined over Val^+. Let $f : Val^+ \rightarrow Val^+$ be defined $f(w) = \tau$ if $w \in Val$ and $f(w) = w$ otherwise. We also define $g(w) = w'$ if $w \in Val$ and $g(w) = w$ otherwise. Let $+$ denote binary choice and let D be the constant with associated definition

$$D \Longleftarrow x \in Val?(a!\mathbf{O} + g(x)!\mathbf{O} + \tau!D) + \sum_{v \in Val} v!(b!\mathbf{O} + g(v)!\mathbf{O} + \tau!D)$$

and let $C[\_]$ be the context $(\_|D)_{(f,Id)}$. Note that we do not require that g be in the language *ValExp* as we can simply use $(x!\mathbf{O})_{(g,Id)}$ in place of $g(x)!\mathbf{O}$.

Let $S = \{(p,q) \mid C[p] \sim_{barb} C[q], p,q : Val\}$, where $p : Val$ means that p is a closed term defined over the value set Val. We know that $(p_0, q_0) \in S$ by hypothesis, so we aim to show that S is a noisy bisimulation.

Suppose that $p \xrightarrow{v_0!} p'$. Then $C[p] \xrightarrow{\tau!} C'_{v_0}[p']$, where

$$C'_{v_0}[\_] = (\_|(a!\mathbf{O} + g(v_0)!\mathbf{O} + \tau!D))_{(f,Id)}.$$

We know that $C[p] \sim_{barb} C[q]$ so $C[q] \xrightarrow{\tau!} r$ for some $r \sim_{barb} C'_{v_0}[p']$. Now $C'_{v_0}[p'] \downarrow a$ so $r \downarrow a$ necessarily, thus $q \xrightarrow{v_1!} q'$ and $r \equiv C'_{v_1}[q']$ for some v_1, q'. We also know that $C'_{v_0}[p'] \downarrow v'_0$. So it must be that $r \downarrow v'_0$, which forces $v_1 = v_0$. We have that $q \xrightarrow{v_0!} q'$ and must now show that $(p',q') \in S$.

Observe that $C'_{v_0}[p'] \xrightarrow{\tau!} C[p']$. It must be the case that $r \xrightarrow{\tau!} r'$ with $r' \sim_{barb} C[p']$ because $r \sim_{barb} C'_{v_0}[p']$. But $C[p'] \not\downarrow v$ for any v so $r' \downarrow v$ cannot hold for any v. Thus $r' \equiv C[q']$, that is $(p',q') \in S$.

Suppose $p \xrightarrow{v_0?} p'$. Then $C[p] \xrightarrow{\tau!} C''_{v_0}[p']$, where

$$C''_{v_0}[\_] = (\_|(b!\mathbf{O} + g(v_0)!\mathbf{O} + \tau!D))_{(f,Id)}.$$

We have $C[q] \xrightarrow{\tau!} r$ for some $r \sim_{barb} C''_{v_0}[p']$. Now $C''_{v_0}[p'] \downarrow b$, so $r \downarrow b$. This means $r \equiv C''_{v_1}[q']$ for some v_1, q', such that $q \xrightarrow{v_1??} q'$. We know $C''_{v_0}[p'] \downarrow v'_0$. So it must be that $r \downarrow v'_0$, which forces $v_1 = v_0$. We have $q \xrightarrow{v_0??} q'$ and must show that $(p',q') \in S$.

It is clear that $C''_{v_0}[p'] \xrightarrow{\tau!} C[p']$. So there exists an r' such that $r \xrightarrow{\tau!} r'$ with $r' \sim_{barb} C[p']$. Now $C[p] \not\downarrow v$ for any v, so it is also the case that $r' \not\downarrow v$ for all v. Thus $r' \equiv C[q']$, which means that $(p',q') \in S$.

Suppose $p \xrightarrow{\tau!} p'$. Then $C[p] \xrightarrow{\tau!} C[p']$, so $C[q] \xrightarrow{\tau!} r$ for some $r \sim_{barb} C[p']$. But $C[p'] \not\downarrow a, b$, so it is also the case that $r \not\downarrow a, b$. This means that no communication can have taken place between the context and q. Thus $q \xrightarrow{\tau!} q'$ with $r \equiv C[q']$, so $(p',q') \in S$. ■

Although noisy bisimulation equivalence is not preserved by choice this can easily be taken into account.

Definition 3.1.5 *Let $p \simeq_n q$ if (omitting symmetric clauses for q)*

- *whenever $p \xrightarrow{w!} p'$ then $q \xrightarrow{w!} q'$ for some q' such that $p' \sim_n q'$*
- *whenever $p \xrightarrow{v?} p'$ then $q \xrightarrow{v?} q'$ for some q' such that $p' \sim_n q'$.*

We say that p and q are noisy congruent.

Theorem 3.1.6 *$(p \sim^c_{barb} q) : Val^+$ if and only if $(p \simeq_n q) : Val$.*

Proof It is straightforward to adapt Proposition 3.1.3 to show that $\simeq_n$ is preserved by all CBS operators. Since $(p \simeq_n q) : Val$ trivially implies that $(p \sim_{barb} q) : Val^+$ it follows immediately that $(p \simeq_n q) : Val$ implies $(p \sim^c_{barb} q) : Val^+$.

Conversely, suppose $(p \sim^c_{barb} q) : Val^+$. Let a be the new value, not occurring in Val used to define Val^+. Then $p + x \in Val?a!\mathbf{O} \sim_n q + x \in Val?a!\mathbf{O}$ implies $(p \simeq_n q) : Val$. But $(p \sim^c_{barb} q) : Val^+$ tells us that for every static context $C[\,]$ over Val^+, $C[p + x \in Val?a!\mathbf{O}] \sim_{barb} C[q + x \in Val?a!\mathbf{O}]$ and therefore by Proposition 3.1.4 it follows that $p + x \in Val?a!\mathbf{O} \sim_n q + x \in Val?a!\mathbf{O}$. ■

This theorem justifies our choice of $\simeq_n$ as the appropriate version of strong bisimulation equivalence for CBS. For this reason, the relation $\simeq_n$ will be studied in the next few sections.

3.2 Characterising strong noisy congruence over simple agents

Our intention is to give an axiomatic characterisation of strong noisy congruence over the finite sublanguage of CBS. Finite CBS is the sub-language of CBS where summation is restricted to be finite and agent declarations for recursion are not used. We will pursue this goal in three stages: firstly we will cut down the finite language to a very restricted class of simple agents $\mathcal{SA}$ in which there is no parallelism, no translation functions and no pattern matching on the input prefixes. A suitable characterisation is given for this language which provides a basic understanding of how noisy and strong bisimulation differ. Then the pattern matching facility is accounted for by extending the class $\mathcal{SA}$ to simple patterned agents $\mathcal{SPA}$ and we show how to integrate this feature with the previous characterisation. Finally, we describe an expansion theorem for the CBS parallel operator and give codings for the translation functions to map agents of finite CBS into $\mathcal{SPA}$.

So, we describe the class $\mathcal{SA}$ by the following grammar:

$$t ::= \mathbf{O} \mid e!t \mid x?t \mid b \gg t \mid t + t.$$

The syntax $x?t$ is shorthand for $x \in Val?t$ and we use binary summation. We will often write $\sum_I t_i$ for nested binary sums of all the agents t_i. Unlike CBS, processes in $\mathcal{SA}$ have the very simple property that if they can discard one value then they can discard every value, or equivalently if they can input one value they can input every value:

Lemma 3.2.1 *For all processes p in $\mathcal{SA}$ if there exists some value v such that $p \xrightarrow{v:}$ then for every value v, $p \xrightarrow{v:}$.*

Proof By structural induction on p. ■

This property will prove invaluable in developing the axiomatisation of noisy congruence over $\mathcal{SA}$. For convenience let us introduce the notation $p \xrightarrow{:}$ to denote the fact that p can discard.

Consider the four axioms required to characterise strong bisimulation equivalence over CCS terms

$$\begin{aligned} X + \mathbf{O} &= X \\ X + X &= X \\ X + Y &= Y + X \\ (X + Y) + Z &= X + (Y + Z). \end{aligned}$$

Noisy bisimulation strictly contains strong bisimulation so we will require at least these four axioms in order to characterise the noisy equivalence. Call these axioms $\mathcal{A}$.

We ask what further properties of noisy bisimulation equivalence fail to be captured by the axioms of $\mathcal{A}$. Recall that noisy equivalence reflects the fact that the reception and discard of a

EQUIV $\dfrac{}{p=p} \quad \dfrac{p=q}{q=p} \quad \dfrac{p=q \quad q=r}{p=r}$

AXIOM $\dfrac{p=q \in \text{ Axioms}}{p=q}$

CONG $\dfrac{p_1=q_1 \quad p_2=q_2}{p_1+p_2=q_1+q_2}$

α-CONV $\dfrac{}{x?t=y?t[y/x]} \quad y \notin fv(t)$

cl-INPUT $\dfrac{\sum_{i\in I}\tau!t_i[v/x]=\sum_{j\in J}\tau!u_j[v/x] \quad \text{for every } v \in Val}{\sum_{i\in I}x?t_i=\sum_{j\in J}x?u_j}$

OUTPUT $\dfrac{p=q, [\![e]\!]=[\![e']\!]}{e!p=e'!q}$

BOOL $\dfrac{[\![b]\!]=\mathbf{tt}}{b \gg p=p} \quad \dfrac{[\![b]\!]=\mathbf{ff}}{b \gg p=\mathbf{O}}$

Figure 3.2. Inference Rules

value by a process p is considered to be identical providing that the future behaviour of p does not depend upon the receipt of the value. This means, for example, that

$$v!p+x?v!p \sim_n v!p$$

for any process p. Note that p is closed so x does not occur freely in $v!p$ and therefore cannot affect the future behaviour of the term $v!p$. Indeed if q is any process which can discard, *i.e.* $q \xrightarrow{:}$, then

$$q+x?q \sim_n q$$

because q can discard any value. This in turn means that

$$w!(q+x?q) \simeq_n w!q.$$

The content of this observation can be captured by a new axiom schema, *cl-Noisy* :

$$\boxed{\textit{cl-Noisy}: \quad w!(p+x?p)=w!p \quad \text{if } p \xrightarrow{:}}$$

The side-condition $p \xrightarrow{:}$ can be replaced by a syntactic condition that p is of the form

$$\sum_{i\in I} b_i \gg e_i!p_i$$

for some finite index set I. We will use the meta-variable $p_!$ for such a p. Let $\mathcal{A}_{\mathcal{N}}$ be the set of equations $\mathcal{A}$ together with the axiom schema *cl-Noisy* . The inference rules required to support these axioms by allowing them to be applied within contexts are listed in Figure 3.2. CBS is a value-passing algebra and as such demands more powerful rules than the ability to substitute equals for equals within context, *cf.* rules OUTPUT, BOOL, cl-INPUT. We would like to draw attention to the rule for inferring input prefixing, cl-INPUT. It is not generally possible to infer

$x?t = x?u$ from the hypothesis $t = u$ because the proof rules only allow manipulation of closed terms. An obvious approach to remedy this might be to introduce an infinitary rule, as suggested by [45],

$$\frac{t[v/x] = u[v/x] \quad \text{for every } v \in Val}{x?t = x?u}.$$

But this rule, although sound, isn't powerful enough for early operational semantics. See [41] for further details.

So, rather than considering the congruence generated by a set of axioms *Axiom*, we consider the identities derivable in the proof system given in Figure 3.2. For any agents p, q let $\mathcal{A}_{\mathcal{N}} \vdash_{cl} p = q$ mean that $p = q$ can be derived in this proof system from the axioms $\mathcal{A}_{\mathcal{N}}$, *i.e.* the axioms $\mathcal{A}$ and the axiom schema *cl-Noisy* .

Theorem 3.2.2 *(Soundness and Completeness) For all agents $p, q \in \mathcal{SA}$, $\mathcal{A}_{\mathcal{N}} \vdash_{cl} p = q$ if and only if $p \simeq_n q$.*

We omit the proof of this theorem as it can be reconstructed from those of Theorem 3.2.5 and Theorem 3.2.12.

3.2.1 A characterisation over open terms

In this section the symbolic approach comes to the fore as we show how the need for an infinitary rule of inference in the proof system above may be obviated by developing a proof system for open terms. The rule for inferring $x?t = x?u$ could then require the simple hypothesis that $t = u$ except that the presence of boolean conditionals in the language complicate judgements about open terms slightly. Our sequents now will be of the form

$$b \triangleright t = u$$

where b is a boolean expression. The idea is that a sequent $t = u$ would abstractly represent the collection of seqents $t\delta = u\delta$ for all data environments δ. The sequent $b \triangleright t = u$ decorated with a boolean similarly represents the collection of sequents $t\delta = u\delta$ where δ is such that $\delta \models b$. As an example, consider booleans c_1, c_2, d_1, d_2 such that $c_1 \vee c_2 = d_1 \vee d_2$. Then the collection of sequents $t\delta = u\delta$ represented by the abstract sequents $c_1 \triangleright t = u$ and $c_2 \triangleright t = u$ are in fact exactly the same collection of sequents represented by $d_1 \triangleright t = u$ and $d_2 \triangleright t = u$ but differently grouped. Thus booleans are a convenient method of describing ways of partitioning collections of sequents. This partitioning corresponds to the ability to perform case analysis in the proof system.

We adapt the inference rules of Figure 3.2 to incorporate the new form of sequent and include various structural rules for manipulating the booleans guarding sequents. These rules, listed in Figure 3.3, are almost identical to those in [41] save for small notational differences. The structural rules for booleans allow the following intuitive properties of sequents to be derived.

Proposition 3.2.3

(i) $b \models b'$ *implies* $\vdash b \triangleright t = b' \gg t$

(ii) $\vdash b \gg (t + u) = (b \gg t) + (b \gg u)$

(iii) $\vdash (b \gg t) + (b' \gg t) = b \vee b' \gg t$

(iv) $\vdash b \gg (b' \gg t) = b \wedge b' \gg t$

(v) $\vdash t = t + b \gg t$

(vi) $\vdash b \wedge b' \triangleright t = u$ *implies* $\vdash b \triangleright b' \gg t = b' \gg u$.

EQUIV $$\frac{}{\mathbf{tt} \rhd t = t} \qquad \frac{b \rhd t = u}{b \rhd u = t} \qquad \frac{b \rhd t = u \quad b \rhd u = v}{b \rhd t = v}$$

AXIOM $$\frac{t = u \in \text{Axioms}}{\mathbf{tt} \rhd t = u}$$

CONG $$\frac{b \rhd t_1 = u_1 \quad b \rhd t_2 = u_2}{b \rhd t_1 + t_2 = u_1 + u_2}$$

α-CONV $$\frac{}{\mathbf{tt} \rhd x?t = y?t[y/x]} \quad y \notin fv(t)$$

INPUT $$\frac{b \rhd \sum_{i \in I} \tau!t_i = \sum_{j \in J} \tau!u_j}{b \rhd \sum_{i \in I} x?t_i = \sum_{j \in J} x?u_j} \quad x \notin fv(b)$$

OUTPUT $$\frac{b \models e = e' \quad b \rhd t = u}{b \rhd e!t = e'!u}$$

TAU $$\frac{b \rhd t = u}{b \rhd \tau!t = \tau!u}$$

GUARD $$\frac{b \wedge b' \rhd t = u \quad b \wedge \neg b' \rhd \mathbf{0} = u}{b \rhd b' \gg t = u}$$

CASE $$\frac{b \models b_1 \vee b_2 \quad b_1 \rhd t = u \quad b_2 \rhd t = u}{b \rhd t = u}$$

ABSURD $$\frac{}{\mathbf{ff} \rhd t = u}$$

Figure 3.3. Inference Rules

Proof See [41]. ■

Using the modified proof system for open terms we now show that the axioms $\mathcal{A}$, along with a generalisation of axiom schema *cl-Noisy* to open terms, provide a sound and complete axiomatisation for strong noisy congruence over $\mathcal{SA}$. The generalisation of *cl-Noisy* is

$$\boxed{Noisy: \quad e!(t_! + x?t_!) = e!t_! \quad \text{if } x \notin fv(t_!)}$$

where $t_!$ is a term of the form

$$\sum_{i \in I} b_i \gg e_i!t_i.$$

Note that any closed instantiation of such a term discards every transmitted value since it can not receive an input. Allowing a slight abuse of notation let us again use $\mathcal{A}_{\mathcal{N}}$ to refer to the axioms $\mathcal{A}$ along with the generalised axiom *Noisy* . We also write $\mathcal{A}_{\mathcal{N}} \vdash b \rhd t = u$ to mean that $b \rhd t = u$ can be derived in the proof system of Figure 3.3 from the axioms in $\mathcal{A}_{\mathcal{N}}$.

Lemma 3.2.4 *(Axiom Noisy is sound) For all* δ*, if* $x \notin fv(t_!)$ *then* $(e!(t_! + x?t_!))\delta \simeq_n (e!t_!)\delta$.

Proof Consider an arbitrary closed instantiation of *Noisy* : $w!(p + x?p) \simeq_n w!p$ and p has the form $p_!$. It is sufficient to show that $p + x?p \sim_n p$. Let I be the identity relation over agents. We show that $I' = I \cup \{(p + x?p, p)\}$ is a noisy bisimulation. The only non-trivial move to match is $p + x?p \xrightarrow{v?} p[v/x]$. Since $x \notin fv(p)$ the agent $p[v/x]$ coincides with p. Also since p is $p_!$ we know that $p \xrightarrow{v:} p$, which is the required match for $p + x?p \xrightarrow{v?} p[v/x]$. ■

Proposition 3.2.5 *(Soundness) If* $\mathcal{A}_{\mathcal{N}} \vdash b \rhd t = u$ *and* $\delta \models b$ *then* $t\delta \simeq_n u\delta$.

Proof It is sufficient to check that all of the individual rules and axioms are sound. This is straightforward and the only novelty is the axiom schema *Noisy* which is treated in the previous Lemma. ■

The converse of this Proposition, completeness, is much harder to prove. In fact this will be our first demonstration of the use of symbolic semantics and symbolic bisimulations. We proceed by giving a symbolic operational semantics for the class of agents $\mathcal{SA}$ and these naturally define a substitution saturated symbolic graph for $\mathcal{SA}$ in the same way that Figure 2.3 defines a saturated symbolic graph for value-passing CCS. The graph for CBS has a single channel name, which we omit, and a single neutral action : which denotes discard. We can then define noisy symbolic bisimulation $\sim_n^b$ on this graph. We show an analogue of Proposition 2.5.4 for noisy congruence. Using this result, completeness of the proof system for open terms is then expressed as

$$t \simeq_n^b u \text{ implies } \mathcal{A}_{\mathcal{N}} \vdash b \rhd t = u.$$

We present the inference rules used to generate the symbolic transition relation $\longmapsto$ in Figure 3.4. Symmetric rules for the choice operator have been omitted. Note that in the transition $\overset{b,\alpha}{\longmapsto}$, α has the form $:, x?$ or $e!$.

Proposition 3.2.6

(i) *if* $t\delta \xrightarrow{\tau!} q$ *then* $\exists b, t' \cdot t \overset{b,\tau!}{\longmapsto} t'$ *where* $\delta \models b, q \equiv t'\delta$ *and conversely if* $t \overset{b,\tau!}{\longmapsto} t'$ *and* $\delta \models b$ *then* $\exists q \cdot t\delta \xrightarrow{\tau!} q$ *and* $q \equiv t'\delta$

(ii) *if* $t\delta \xrightarrow{v!} q$ *then* $\exists b, e, t' \cdot t \overset{b,e!}{\longmapsto} t'$ *where* $\delta \models b, [[e]]\delta = v, q \equiv t'\delta$ *and conversely if* $t \overset{b,e!}{\longmapsto} t'$ *and* $\delta \models b, [[e]]\delta = v$ *then* $\exists q \cdot t\delta \xrightarrow{v!} q$ *and* $q \equiv t'\delta$

(iii) *if* $t\delta \xrightarrow{v?} q$ *then* $\exists b, x, t' \cdot t \overset{b,x?}{\longmapsto} t'$ *where* $\delta \models b, q \equiv t'\delta[v/x]$ *and conversely if* $t \overset{b,x?}{\longmapsto} t'$ *where* $\delta \models b$ *then* $\exists q \cdot t\delta \xrightarrow{v?} q$ *and* $q \equiv t'\delta[v/x]$

Discard	Input	Output
$\mathbf{0} \overset{\mathbf{tt},:}{\longmapsto} \mathbf{0}$		
	$\dfrac{}{x?t \overset{\mathbf{tt},x?}{\longmapsto} t}$	
$e!t \overset{\mathbf{tt},:}{\longmapsto} e!t$		$e!t \overset{\mathbf{tt},e!}{\longmapsto} t$
$\dfrac{t \overset{b,:}{\longmapsto} t \quad u \overset{b',:}{\longmapsto} u}{t+u \overset{b'\wedge b,:}{\longmapsto} t+u}$	$\dfrac{t \overset{b,x?}{\longmapsto} t'}{t+u \overset{b,x?}{\longmapsto} t'}$	$\dfrac{t \overset{b,e!}{\longmapsto} t'}{t+u \overset{b,e!}{\longmapsto} t'}$
$b' \gg t \overset{\neg b',:}{\longmapsto} b' \gg t$		
$\dfrac{t \overset{b,:}{\longmapsto} t}{b' \gg t \overset{b,:}{\longmapsto} b' \gg t}$	$\dfrac{t \overset{b,x?}{\longmapsto} t'}{b' \gg t \overset{b'\wedge b,x?}{\longmapsto} t'}$	$\dfrac{t \overset{b,e!}{\longmapsto} t'}{b' \gg t \overset{b'\wedge b,e!}{\longmapsto} t'}$

Figure 3.4. Abstract operational semantics

(iv) $t\delta \xrightarrow{v:} t\delta$ *if and only if* $\exists b \cdot \delta \models b$ *and* $t \overset{b,:}{\longmapsto} t$

Proof A minor variation on Lemma 3.2 of [41]. ■

Let $\mathbf{S} = \{S^b \mid b \in BoolExp\}$ be a boolean indexed family of relations. Define $\mathcal{NSB}(\mathbf{S})$ to be the family of relations such that

$(t,u) \in \mathcal{NSB}(\mathbf{S})^b$ if whenever $t \overset{b_1,\alpha}{\longmapsto} t'$ ($\alpha \equiv x?$ or $e!$) there exists a variable z such that $z \notin fv(b,t,u)$ and a $b \wedge b_1$-*partition*, B, and for each $b' \in B$ there exists a $u \overset{b_2,\beta}{\longmapsto} u'$ such that $b' \models b_2$ and

- if α is $e!$ then $\beta \equiv e'!$ with $b' \models e = e'$ and $(t',u') \in S^{b'}$
- if α is $x?$ then $\beta \equiv y?$ for some y and $(t'[z/x], u'[z/y]) \in S^{b'}$ or $\beta \equiv :$ and $(t'[z/x], u') \in S^{b'}$

(Symmetric condition on transitions from u omitted).

We call $\mathbf{S}$ a ***noisy symbolic bisimulation*** if $\mathbf{S} \subseteq \mathcal{NSB}(\mathbf{S})$ (point-wise inclusion) and denote the largest such relation by $\{\sim_n^b\}$. It is evident that this relation is not a congruence for the language as it is not preserved by summation. As before we modify it so that we obtain the largest *CBS* congruence contained within it:

Let $t \simeq_n^b u$ if whenever $t \overset{b_1,\alpha}{\longmapsto} t'$ ($\alpha \equiv x?$ or $e!$) there exists a variable z such that $z \notin fv(b,t,u)$ and a $b \wedge b_1$-*partition*, B, and for each $b' \in B$ there exists a $u \overset{b_2,\beta}{\longmapsto} u'$ such that $b' \models b_2$ and

- if α is $e!$ then $\beta \equiv e'!$ with $b' \models e = e'$ and $t' \sim_n^{b'} u'$
- if α is $x?$ then $\beta \equiv y?$ for some y and $t'[z/x] \sim_n^{b'} u'[z/y]$

(Symmetric condition on transitions from u omitted).

Theorem 3.2.7 $t \simeq_n^b u$ *if and only if* $\forall \delta \cdot \delta \models b$ *implies* $t\delta \simeq_n u\delta$. *In particular for agents we have that* $p \simeq_n^{tt} q$ *if and only if* $p \simeq_n q$

Proof As in [41], we use Proposition 3.2.6 to prove that whenever $\mathbf{S}$ is a noisy symbolic bisimulation then

$$R_{\mathbf{S}} \overset{def}{=} \left\{(t\delta, u\delta) \mid \exists b \cdot \delta \models b \text{ and } (t,u) \in S^b\right\}$$

is a noisy bisimulation. Similarly, whenever $\mathcal{R}$ is a noisy bisimulation then

$$S^b_{\mathcal{R}} \overset{def}{=} \{(t,u) \mid \delta \models b \text{ implies } (t\delta, u\delta) \in \mathcal{R}\}$$

forms a noisy symbolic bisimulation. The result follows easily from this. We should note here that the proof of this theorem demands a great expressiveness of the boolean metalanguage. Essentially, we require the power to describe given sets of environments. This issue is discussed in [40]. ■

A typical property one requires in order to prove completeness of a proof system is the ability to transform terms into certain syntactic forms. We make use of two types of these syntactic forms: standard forms which allow us to isolate the individual summands of a term, and normal forms which afford a tighter analysis of the boolean guards contained within a term.

Firstly, a term t is said to be in ***standard form*** if it is of the form

$$\sum_{i \in I_!} b_i \gg e_i!t_i + \sum_{i \in I_?} b_i \gg x_i?t_i$$

where $I_!$ and $I_?$ are finite indexing sets. We will call the left hand sum $t_!$ and the right hand sum $t_?$. A term t is called a ***normal form*** if it is of the form

$$\sum_{i \in I} c_i \gg (\sum_{k \in I_i} \alpha_{ik}.t_{ik})$$

where $c_i \wedge c_j \equiv \mathbf{ff}$ whenever $i \neq j$ and $\bigvee_I c_i = \mathbf{tt}$.

An important property of standard/normal forms is that the rearrangement required within a term in order to rewrite it into one of these forms is provable in the proof system. We also require that the *depth* of terms is preserved. We can write this more precisely by defining a ***depth*** function, $d(p)$, on terms. Let $d(p)$ be defined inductively as follows:

- $d(\mathbf{O}) = 0$
- $d(x?t) = d(e!t) = 1 + d(t)$
- $d(b \gg t) = d(t)$
- $d(t_1 + t_2) = \max\{d(t_1), d(t_2)\}$

Lemma 3.2.8 *For every term t, there exist standard and normal forms $sf(t), nf(t)$ (respectively) such that $d(t) = d(sf(t) = d(nf(t))$ and*

$$\mathcal{A} \vdash t = sf(t) \textit{ and } \mathcal{A} \vdash t = nf(t).$$

Proof It is easy to see that every term can be provably transformed into a standard form by using Proposition 3.2.3, part (ii) to distribute booleans across sums and using axioms $\mathcal{A}$ to remove occurrences of $\mathbf{O}$. So, let the $sf(t)$ be $\sum_{j \in J} b_j \gg \alpha_j.t_J$. For each $K \subseteq J$ we define c_K to be the boolean expression $\bigwedge_{k \in K} b_k \wedge \bigwedge_{k' \in J-K} \neg b_{k'}$. Thus we have $\bigvee c_K = \mathbf{tt}, c_K \wedge c_{K'} = \mathbf{ff}$ whenever $K \neq K'$. Using Proposition 3.2.3 we can show

$$\vdash \mathbf{tt} \rhd t = \sum_{K} c_K \gg (\sum_{k \in K} b_k \gg \alpha_k.t_k).$$

Using CASE and Proposition 3.2.3 we can obtain, for each K,

$$\vdash c_K \rhd t = \sum_K c_K \gg (\sum_{k\in K} \alpha_k.t_k).$$

Thus, given that $\bigvee c_K = \mathbf{tt}$, CASE gives

$$\vdash \mathbf{tt} \rhd t = \sum_K c_K \gg (\sum_{k\in K} \alpha_k.t_k).$$

It is clear that the depth of the term t is unchanged as a result of these transformations. ■

As an example of how the mutually exclusive guards of normal forms can be useful we refer the reader to [41], Proposition 3.7 and note that the proof there can also be used to conclude that

$$\text{INPUT}^{\gg} \quad \frac{b \rhd \sum_{i\in I} c_i \gg \tau!t_i = \sum_{j\in J} d_j \gg \tau!u_j}{b \rhd \sum_{i\in I} c_i \gg x?t_i = \sum_{j\in J} d_j \gg x?u_j}$$

(where $x \notin fv(b, c_i, d_j)$) is a derived rule of the proof system.

Given a standard form $t \equiv \sum_{i\in I} b_i \gg \alpha_i.t_i$ we notice that we can give boolean conditions to describe when t can input or discard. For instance, we know that the boolean b will guarantee that t has the ability to receive a value if $b \models \bigvee_{I_?} b_i$. Similarly, b will guarantee a discard move from t if $b \models \bigwedge_{I_?} \neg b_i$. This motivates the following definition.

For an arbitrary term t, the ***discard condition***, $DC(t)$ is the weakest condition under which t is triggered to discard, *i.e.* $t \overset{DC(t),:}{\longmapsto} t$ and whenever $t \overset{b,:}{\longmapsto} t$ then $b \models DC(t)$. We can give syntactic descriptions of $DC(t)$ whenever t is a standard or a normal form.

Lemma 3.2.9 *Let t be a standard form $\sum_{i\in I} b_i \gg \alpha_i.t_i$, and let $t' \equiv \sum_{K\in\mathcal{P}I} c_K \gg (\sum_{i\in K} \alpha_{K_i}.t_{K_i})$ be the normal form constructed from t as described in Lemma 3.2.8. Then*

$$DC(t) = \bigwedge_{i\in I_?} \neg b_i = \bigwedge_{?(K)} \neg c_K$$

where $?(K)$ holds if there exists an $i \in K$ such that α_{K_i} is of the form $x?$ for some x.

Proof Let b denote $\bigwedge_{i\in I_?} \neg b_i$ and let c denote $\bigwedge_{?(K)} \neg c_K$. It is easy to see from the symbolic operational semantics that $t \overset{b,:}{\longmapsto} t$ so we immediately have that $b \models DC(t)$ by definition.

Conversely, suppose $\delta \models DC(t)$. We know $t \overset{DC(t),:}{\longmapsto} t$ so, by Proposition 3.2.6, we know that $t\delta \xrightarrow{v:} t\delta$ for any v. Suppose $\delta \models b_i$ for some $i \in I_?$. Then we have a contradiction because $t \overset{b_i,x?}{\longmapsto} t'$ for some x, whence $t\delta \xrightarrow{v?} q$ for some v, q. Thus $\delta \not\models b_i$ for all $i \in I_?$, that is to say $\delta \models \bigwedge_{i\in I_?} \neg b_i$.

To show the latter equality we first observe that $\delta \models b$ implies $\delta \not\models (\bigwedge_{j\in K} b_j) \wedge (\bigwedge_{j\in I-K} \neg b_j)$ whenever $I_? \cap K \neq \emptyset$. This amounts to saying that $\delta \not\models c_K$ whenever $?(K)$. That is, $\delta \models \neg c_K$ for all K such that $?(K)$ and so $\delta \models c$.

Conversely, suppose that $\delta \models c$ and suppose for contradiction that there is an $i_0 \in I_?$ such that $\delta \models b_{i_0}$. We let $K_0 = \{i_0\}$ and define a strictly increasing sequence of subsets of I

$$K_0 \subset K_1 \subset \cdots \subset K_n \subset \cdots$$

with the property that $\delta \models b_j$ for all $j \in K_n$ for all n. Given K_n we know that $?(K_n)$ because $i_0 \in K_n$. We know then that $\delta \models \neg c_{K_n}$ because $\delta \models c$. Recall that $\neg c_{K_n} = (\bigvee_{j\in K_n} \neg b_j) \vee (\bigvee_{j\in I-K_n} b_j)$. We know that there must exist a $j_0 \in I - K_n$ such that $\delta \models b_{j_0}$ because of the property of K_n that $\delta \models b_j$ for each $j \in K_n$. Let $K_{n+1} = K_n \cup \{j_0\}$. This defines a strictly increasing infinite sequence which is bounded by the finite set I, which is a contradiction. Thus $\delta \not\models b_i$ for each $i \in I_?$, hence $\delta \models b$. ■

We come now to the theorem which lies at the heart of the completeness theorem. It relates the noisy symbolic bisimulation relation to the noisy symbolic congruence relation. The completeness theorem for finite CCS terms with respect to weak bisimulation congruence $\approx_c$, [74], page 156, relies on a similar relationship between weak bisimulation, $\approx$, and bisimulation congruence, $\approx_c$: if $p \approx q$ then either $p \approx_c q$, $p \approx_c \tau.q$ or $\tau.p \approx_c q$. For CBS the corresponding relation is if $p \sim_n q$ then either $p \simeq_n q$, $p \simeq_n x?q + q$ or $x?p + p \simeq_n q$. However, at the symbolic level the relationship is a little more complicated. We first express this relationship between normal forms and then use Lemma 3.2.9 to discover the relationship for standard terms.

Theorem 3.2.10 *If t, u are normal forms then $t \sim_n^b u$ if and only if there exists a b-partition, B, such that for $x \notin fv(t,u,b')$ for each $b' \in B$ one of the following holds:*

1. $(t \simeq_n^{b'} u)$
2. $(t \simeq_n^{b'} u + x?u)$ *and* $b' \models DC(u)$
3. $(t + x?t \simeq_n^{b'} u)$ *and* $b' \models DC(t)$.

Proof The '$\Leftarrow$' direction is quite simple to prove using Theorem 3.2.7 so we concentrate on the '$\Rightarrow$' direction. One approach to proving this would be to prove the corresponding result about closed terms and then use Theorem 3.2.7 to translate to open terms. A more illuminating direct approach is given here.

We have normal forms for t and u, that is, $t \equiv \sum_{i\in I} c_i \gg (\sum_{k\in I_i} \alpha_{ik}.t_{ik})$ and $u \equiv \sum_{j\in J} d_j \gg (\sum_{l\in J_j} \beta_{jl}.u_{jl})$.

Let $B' \overset{def}{=} \{b \wedge c_i \wedge d_j \mid i \in I, j \in J\}$. Then we know that $\bigvee B' = b$. Consider $b' \equiv b \wedge c_i \wedge d_j \in B'$. We know that $t \sim_n^{b'} u$ because $b' \models b$. So whenever $t \overset{c_i,x?}{\longmapsto} t_k$, there exists a b'-partition, B_k such that for each $b_{k_i} \in B_k$ there is a matching move from u. Similarly, there is a b'-partition, B_l for each move $u \overset{d_j,x?}{\longmapsto} u_l$. We have a set of n partitions $\{B_{k_1}, B_{k_2}, \ldots, B_{k_n}\}$ and m partitions $\{B_{l_1}, B_{l_2}, \ldots, B_{l_m}\}$, say. If $n = m = 0$ then we define $B_{b'}$ to be $\{b'\}$. Otherwise we consider all conjunctions of length $n+m$ whose conjucts are drawn one from each partition. Define $B_{b'} = \left\{ (\bigwedge_{i=1}^{n} b_i) \wedge (\bigwedge_{j=1}^{m} b_j) \mid b_i \in B_{k_i}, b_j \in B_{l_j} \right\}$. Then $\bigvee B_{b'} = b'$ and furthermore $B_{b'}$ enjoys the following property:

For each $b'' \in B_{b'}$ we have that $t \sim_n^{b''} u$ and whenever $t \overset{c_i,x?}{\longmapsto} t'$ then there is a u' such that $u \overset{d_j,x?}{\longmapsto} u'$ with $t' \sim_n^{b''} u'$ and $b'' \models d_j$ or $u \overset{DC(u),:}{\longmapsto} u$ with $t' \sim_n^{b''} u$ and $b'' \models DC(u)$. Similarly for u.

So we let $B = \bigcup_{b' \in B'} B_{b'}$ and consider the three cases which arise. Take $b'' \in B_{b'}$.

Case 1 There exists a $t \overset{c_i,x?}{\longmapsto} t'$ such that for all $u \overset{d_j,x?}{\longmapsto} u_{jl}$, $t' \not\sim_n^{b''} u_{jl}$.

Therefore $u \overset{DC(u),:}{\longmapsto} u$ with $t' \sim_n^{b''} u$ and $b'' \models DC(u)$ since $t \sim_n^{b''} u$. We now show that $t \simeq_n^{b''} u + x?u$.

Recall that b'' corresponds to just one of the c_i and d_j, in the sense that $b'' \wedge c_{i'} = b'' \wedge d_{j'} = \mathbf{ff}$ whenever $i' \neq i, j' \neq j$. Therefore we need only consider moves of the form $t \overset{c_i,\alpha}{\longmapsto}$ and $u \overset{d_j,\alpha}{\longmapsto}$.

Suppose then that $t \overset{c_i,e!}{\longmapsto} t_{ik}$. Since $t \sim_n^{b''} u$ we know there exists a b''-partition, B'', such that for each $b_1 \in B''$ there exists a $u \overset{d_j,e'!,u}{\longmapsto}{}_{jl}$ with $b_1 \models d_j$, $b_1 \models (e = e')$ and $t_{ik} \sim_n^{b_1} u_{jl}$. This means $u + x?u \overset{d_j,e'!}{\longmapsto} u_{jl}$ also so we have a match for $t \overset{c_i,e!}{\longmapsto} t_{ik}$.

Suppose $t \overset{c_i,x?}{\longmapsto} t_{ik}$. As $b'' \models d_j$ and $b'' \models DC(u)$ then, by Lemma 3.2.9, $b'' \models \neg d_{j'}$ for each j' such that $?(j')$. Clearly then it cannot be the case that $?(j)$ holds because $b'' \models d_j$. Therefore no $l \in J_j$ and no variable x are such that β_{jl} is $x?$. Thus $u \overset{d_j,x?}{\not\longmapsto}$ and therefore $u \overset{DC(u),:}{\longmapsto} u$ with $t_{ik} \sim_n^{b''} u$. Given this we use b'' to partition itself, $u + x?u \overset{\mathbf{tt},x?}{\longmapsto} u$ being the matching move.

Suppose that $u + x?u \overset{d_j,e!}{\longmapsto} u_{jl}$. Then, as before, we use the fact that $t \sim_n^{b''} u$ to get a matching partition and move. Suppose that $u + x?u \overset{d,x?}{\longmapsto} u'$. By assumption, d is d_j or $\mathtt{tt}$. Clearly d cannot be d_j because, as we have already established, $u \overset{d_j,x?}{\not\longmapsto} u'$. Thus d must be $\mathtt{tt}$ and u' must be u. Again, b'' partitions itself to get the matching move $t \overset{c_i,x?}{\longmapsto} t_{ik}$.

Case 2 There exists a $u \overset{d_j,x?}{\longmapsto} u'$ such that for all $t \overset{c_i,x?}{\longmapsto} t_{ik}$, $u' \not\sim_n^{b''} t_{ik}$. Symmetrical argument of Case one which yields $t + x?t \simeq_n^{b''} u$ and $b' \models DC(t)$.

Case 3 Neither of the above. That is, for every $t \overset{c_i,x?}{\longmapsto} t_{ik}$ there exists a $u \overset{d_j,x?}{\longmapsto} u_{jl}$ such that $t_{ik} \sim_n^{b''} u_{jl}$. Also, for every $u \overset{d_j,x?}{\longmapsto} u_{jl}$ there exists a $t \overset{c_i,x?}{\longmapsto} t_{ik}$ such that $t_{ik} \sim_n^{b''} u_{jl}$. It is easy to show then that $t \simeq_n^{b''} u$. ■

As a corollary we show that Theorem 3.2.10 can be lifted to deal with the simpler notion of standard form which will be used to prove completeness. We also reinterpret the statement $b' \models DC(t)$ in terms of provability in the proof system.

Corollary 3.2.11 *If t, u are standard forms $\sum_I c_i \gg \alpha_i.t_i$, $\sum_J d_j \gg \beta_j.u_j$ respectively, then $t \sim_n^b u$ if and only if there exists a b-partition, B, such that for $x \notin fv(t,u,b')$ for each $b' \in B$ one of the following holds:*

1. *$(t \simeq_n^{b'} u)$*
2. *$(t \simeq_n^{b'} u + x?u)$ and $\mathcal{A}_{\mathcal{N}} \vdash b' \rhd u = u_!$*
3. *$(t + x?t \simeq_n^{b'} u)$ and $\mathcal{A}_{\mathcal{N}} \vdash b' \rhd t = t_!$*

Proof We construct $nf(t), nf(u)$ as directed in Lemma 3.2.8. We know $t \simeq_n^b nf(t)$ and $u \simeq_n^b nf(u)$ by Soundness. Now apply Theorem 3.2.10 to get the three cases. The first case yields $nf(t) \simeq_n^b nf(u)$ and transitivity gives $t \simeq_n^b u$. Similarly, in the second case we use transitivity and congruence properties of $\simeq_n^b$, moreover we must show $\mathcal{A}_{\mathcal{N}} \vdash b' \rhd u = u_!$. We have $b' \models DC(u)$ so Lemma 3.2.9 tells us that $b' \models \bigwedge_{j \in J_?} \neg d_j$.

It is simple to show that $\mathcal{A}_{\mathcal{N}} \vdash b' \rhd u_! = u_!$. So we need only show $\mathcal{A}_{\mathcal{N}} \vdash b' \rhd u_? = \mathbf{O}$. To do this we show that for each $j \in J_?$ we have $\mathcal{A}_{\mathcal{N}} \vdash b' \rhd d_j \gg x_j?u_j = \mathbf{O}$. This is simply a matter of using ABSURD to get $\mathcal{A}_{\mathcal{N}} \vdash b' \wedge d_j \rhd x_j?u_j = \mathbf{O}$ and then using GUARD to get $\mathcal{A}_{\mathcal{N}} \vdash b' \rhd d_j \gg x_j?u_j = \mathbf{O}$.

The last case can be dealt with similarly. ■

Theorem 3.2.12 *(Completeness) $t \simeq_n^b u$ implies $\mathcal{A}_{\mathcal{N}} \vdash b \rhd t = u$*

Proof By using Lemma 3.2.8 we can assume that t and u are the standard forms $\sum_{i \in I} c_i \gg \alpha_i.t_i$ and $\sum_{j \in J} d_j \gg \beta_j.u_j$ respectively and proceed by induction on $d(t) + d(u)$.

We only show $\mathcal{A}_{\mathcal{N}} \vdash b \rhd t_? = u_?$. The proof of $\mathcal{A}_{\mathcal{N}} \vdash b \rhd t_! = u_!$ is similar and is omitted. Combining both of these we get the required $\mathcal{A}_{\mathcal{N}} \vdash b \rhd t = u$.

Suppose we can prove

$$\mathcal{A}_{\mathcal{N}} \vdash b \wedge c_i \rhd u_? + c_i \gg x_i?t_i = u_?$$

for each $i \in I_?$. Then an application of GUARD will yield

$$\mathcal{A}_{\mathcal{N}} \vdash b \rhd u_? + c_i \gg x_i?t_i = u_?$$

Using CONG we can then combine these to get

$$\mathcal{A}_{\mathcal{N}} \vdash b \rhd u_? + t_? = u_?$$

and an entirely symmetric argument will give us that

$$\mathcal{A}_{\mathcal{N}} \vdash b \rhd t_? = u_? \ (= t_? + u_?).$$

Therefore we only have to fulfil the obligation of showing

$$\mathcal{A}_{\mathcal{N}} \vdash b \wedge c_i \rhd u_? + c_i \gg x_i?t_i = u_?$$

for an arbitrary i.

Let z be a variable not in $fv(b,t,u)$, let $t^z_?$ denote $\sum_{I_?} c_i \gg z?t_i[z/x_i]$ and $t^\tau_?$ denote $\sum_{I_?} c_i \gg \tau!t_i[z/x_i]$. Let $u^z_?, u^\tau_?$ denote the corresponding terms for u. Consider $t^z_? \overset{c_i,z?}{\longmapsto} t_i[z/x_i]$. Since $t \simeq^b_n u$ we know there exists a $b \wedge c_i$-partition, B, such that for each $b' \in B$ there exists a $u^z_? \overset{d_j,z?}{\longmapsto} u_j[z/y_j]$ such that $b' \models d_j$ and $t_i[z/x_i] \sim^{b'}_n u_j[z/y_j]$. We can find standard forms for these terms and then by Theorem 3.2.11 there exists a b'-partition, B' and some fresh variable x such that for each $b'' \in B'$

1 $t_i[z/x_i] \simeq^{b''}_n u_j[z/y_j]$ or

2 $t_i[z/x_i] \simeq^{b''}_n u_j[z/y_j] + x?u_j[z/y_j]$ or

3 $t_i[z/x_i] + x?t_i[z/x_i] \simeq^{b''}_n u_j[z/y_j]$.

In each of these cases we will show how to deduce $\mathcal{A}_{\mathcal{N}} \vdash b'' \rhd \tau!t_i[z/x_i] = \tau!u_j[z/y_j]$.

Case 1 We apply induction and then use the rule TAU.

Case 2 We apply induction again to get

$$\mathcal{A}_{\mathcal{N}} \vdash b'' \rhd t_i[z/x_i] = u_j[z/y_j] + x?u_j[z/y_j]$$

In this case we also know that

$$\mathcal{A}_{\mathcal{N}} \vdash b'' \rhd u_j[z/y_j] = (u_j[z/y_j])_!$$

Using axiom *Noisy* and TAU will then give

$$\mathcal{A}_{\mathcal{N}} \vdash b'' \rhd \tau!t_i[z/x_i] = \tau!u_j[z/y_j]$$

Case 3 Symmetric to case 2.

For each $b'' \in B'$ we have proved $\mathcal{A}_{\mathcal{N}} \vdash b'' \rhd \tau!t_i[z/x_i] = \tau!u_j[z/y_j]$ so we can use CASE to obtain $\mathcal{A}_{\mathcal{N}} \vdash b' \rhd \tau!t_i[z/x_i] = \tau!u_j[z/y_j]$. Given that $b' \models c_i$, $b' \models d_j$ we can use Proposition 3.2.3 and the idempotence axiom of $\mathcal{A}$ to produce

$$\mathcal{A}_{\mathcal{N}} \vdash b' \rhd d_j \gg \tau!u_j[z/y_j] = (c_i \gg \tau!t_i[z/x_i]) + (d_j \gg \tau!u_j[z/y_j]).$$

Adding in the other summands of u we get

$$\mathcal{A}_{\mathcal{N}} \vdash b' \rhd u^\tau_? = u^\tau_? + c_i \gg \tau!t_i[z/x_i]$$

A further application of CASE gives

$$\mathcal{A}_{\mathcal{N}} \vdash b \wedge c_i \rhd u^\tau_? = u^\tau_? + c_i \gg \tau!t_i[z/x_i]$$

Finally, we apply INPUT$^{\gg}$ to get $\mathcal{A}_{\mathcal{N}} \vdash b \wedge c_i \rhd u^z_? = u^z_? + c_i \gg z?t_i[z/x_i]$. The result is obtained by α-conversion. ■

3.3 Agents with pattern-matching on inputs

We now extend our characterisations of strong noisy congruence to a larger class of agents which have the facility to be selective about which values they are prepared to receive. We call this pattern-matching and we recall that this can be described by the syntax $x \in S?$ for input prefixes. The pattern, S, is a set of values not containing τ. We will let $\mathcal{SPA}$ denote this extended class of agents. With the addition of this construct Lemma 3.2.1 is no longer true. For example the agent $x? \in \{0,1\}.\mathbf{O}$ discards the value 3 but it can not discard either of the values 0 or 1.

We are going to show how to adapt the proof systems of Section 3.2 to incorporate this pattern-matching on input prefixes. The main difficulty this presents is that the axiom schema *cl-Noisy* is no longer sufficient. Recall that *cl-Noisy* states that

$$p \simeq_n p + x?p \quad \text{if } p \xrightarrow{:}$$

The condition that $p \xrightarrow{:}$ means that p can discard *every* value which is clearly too strong a condition to demand in the presence of pattern-matching. If we examine the proof of Lemma 3.2.4 then we see that this side-condition on the axiom is required in order to match the move

$$p + x?p \xrightarrow{v?} p$$

for *every* value v. If we only demand that p discards values from the set S then we can no longer expect $p \simeq_n p + x?p$ to hold because we can no longer match the transition above for every v, we can match it for just those v contained in S. Instead we see that $p \simeq_n p + x \in S'?p$, provided S' is contained in S. To express this observation as an axiom we give a syntactic definition of $I(p)$, the set of values which the process p is immediately ready to receive and then, using Lemma 3.1.1, we obtain a description of which values the process p can discard.

The set $I(p)$ is defined inductively as follows:

- $I(\mathbf{O}) = \emptyset$
- $I(e!p) = \emptyset$
- $I(x \in S?t) = S$
- $I(p+q) = I(p) \cup I(q)$
- $I(b \gg p) = \begin{cases} I(p) & \text{if } [\![b]\!] = \mathbf{tt} \\ \emptyset & \text{otherwise} \end{cases}$

Proposition 3.3.1 *For every agent p, $v \in I(p)$ if and only if $p \xrightarrow{v?}$.*

Proof By structural induction on P. ■

The axiom schema *cl-Noisy* is replaced by the schema

$$\boxed{\textit{cl-P-Noisy}: \quad w!(p + x \in S?p) = w!p \qquad \text{if } S \cap I(p) = \emptyset.}$$

Of course, the condition $S \cap I(p) = \emptyset$ merely states, in light of the previous proposition, that S is contained in the set of values which p may discard.

This modification alone is insufficient to capture the behaviour of pattern-matching agents. Consider the two processes $x \in S?t + x \in S'?t$ and $x \in S \cup S'?t$. They are clearly strong noisy congruent, yet there is no way of showing this in the closed term proof system as there are no rules to manipulate pattern sets. In fact we equate these directly with the new axiom schema

$$\boxed{\textit{Pattern}: \quad x \in S?X + x \in S'?X = x \in S \cup S'?X.}$$

We also need the single axiom

$$\boxed{\textit{Empty}: \quad x \in \emptyset?X = \mathbf{O}.}$$

We should note that these additional axioms could be expressed using the single axiom scheme

$$\sum_I x \in S_i?X = x \in \bigcup_I S_i?X$$

and we recognize *Empty* as the nullary version of *Pattern* .

With regards to the inference rules of Figure 3.2, we will obviously need to allow pattern-matching in the rule for α-conversion, *i.e.*

$$\boxed{\alpha\text{-CONV} \quad \frac{}{x \in S?t = y \in S?t[y/x]} \quad \text{if } y \notin fv(t),}$$

and the rule cl-INPUT is replaced with its modified version

$$\boxed{\text{cl-P-INPUT} \quad \frac{t[v/x] = u[v/x] \quad \text{for every } v \in S}{x \in S?t = x \in S?u}.}$$

Notice that, unlike the proof systems for $\mathcal{SA}$, we express this INPUT rule without using summation. With an early semantics it is often the case that without this summation the proof system would fail to be complete. For example, in the proof systems for $\mathcal{SA}$ the agents defined over the naturals, $\mathcal{N}$,

$$x \in \mathcal{N}?(x > 1 \gg p) + x \in \mathcal{N}?(x \leq 1 \gg p)$$

and

$$x \in \mathcal{N}?p + x \in \mathcal{N}?\mathbf{O}$$

can not be proven congruent without the summation on the input rule. However we now have the use of the axiom *Pattern* , wherein lies the gain. In order to prove the above agents congruent we would first use *Pattern* to transform the latter agent into

$$x \in \{0,1\}?p + x \in \{2,3,\ldots\}?p + x \in \{0,1\}?\mathbf{O} + x \in \{2,3,\ldots\}?\mathbf{O},$$

use the modified Input rule, cl-P-INPUT, and then use *Pattern* again.

Let $\mathcal{A}_{\mathcal{P}}$ denote the set of axioms $\mathcal{A}$ augmented with the three axioms: *cl-P-Noisy* , *Pattern* and *Empty* . Use $\mathcal{A}_{\mathcal{P}} \vdash_{cl} p = q$ to denote that $p = q$ can be derived from these axioms using the proof system in Figure 3.2 but with the modified rule for α-conversion and with rule cl-P-INPUT instead of the rule cl-INPUT.

This modified proof system is indeed both sound and complete for strong noisy congruence over $\mathcal{SPA}$ however in order to prove this we must first provide an analogue of Theorem 3.2.10 (The depth of a patterned input is simply $d(x \in S?p) = 1 + d(p)$).

Theorem 3.3.2 *Let $p, q \in \mathcal{SPA}$ then*

$$p \sim_n q \textit{ iff } p + x \in (I(q) - I(p))?p \simeq_n q + x \in (I(p) - I(q))?q.$$

Moreover, when $I(q) - I(p)$ and $I(p) - I(q)$ are both non-empty there exist p', q' such that $d(p') < d(p), d(q') < d(q)$ and $p' \sim_n p \sim_n q \sim_n q'$.

Proof We outline the '$\Rightarrow$' direction. If $p \xrightarrow{v?} p'$ then we know there exists a q' such that $q \xrightarrow{v??} q'$ with $p' \sim_n q'$ because $p \sim_n q$. If $v \in I(q)$ then we know that $q \xrightarrow{v?} q'$. Otherwise $v \notin I(q)$ and $q \xrightarrow{v:} q' (\equiv q)$. In this case though $v \in I(p) - I(q)$ which means that $x \in I(p) - I(q)?q \xrightarrow{v?} q$ matches the move from p.

We know that $x \in I(q) - I(p)?p \xrightarrow{v?} p$ whenever $v \in I(q) - I(p)$. So we require a match from q. $q \xrightarrow{v?} q'$ as $v \in I(q)$ and $v \notin I(p)$ so $p \xrightarrow{v:} p$. Because $p \sim_n q$ we then know that $p \sim_n q'$ necessarily.

When the two sets are both non-empty we let $v_1 \in I(q) - I(p)$, $v_2 \in I(p) - I(q)$, then $p \xrightarrow{v_2?} p'$ for some p' with depth smaller than p. We know $v_2 \notin I(q)$ so $q \xrightarrow{v_2:} q$ must match this move, that is, $p' \sim_n q$. Similarly, we get $q' \sim_n p$ using v_1. Transitivity of $\sim_n$ gives the result. ■

Theorem 3.3.3 *(Soundness and Completeness) For all agents $p, q \in \mathcal{SPA}$*

$$\mathcal{A}_{\mathcal{P}} \vdash_{cl} p = q \textit{ if and only if } p \simeq_n q.$$

Proof The soundness is simply a matter of checking the validity of the axioms and that the new rule preserves the semantic congruence. So we confine our outline to the proof of completeness. Again the proof is by induction on the combined depth of p and q.

Because of the newly introduced axiom *Empty* we can assume that any closed term can be provably transformed to a *patterned* standard form of equal depth, *i.e.* a term of the form

$$\sum_I e_i!p_i + \sum_J x \in S_j?t_j,$$

where each set S_j is non-empty. So let us assume that p and q have the forms

$$\sum_I e_i!p_i + \sum_J x \in S_j?t_j, \quad \sum_K e_k!q_k + \sum_L x \in S_l?u_l$$

respectively. It is sufficient to prove that that

$$\mathcal{A}_{\mathcal{P}} \vdash_{cl} \sum_I e_i!p_i = \sum_K e_k!q_k$$

and

$$\mathcal{A}_{\mathcal{P}} \vdash_{cl} \sum_J x \in S_j?t_j = \sum_L x \in S_l?u_l$$

and as an example we consider the latter. To establish this it is sufficient, by symmetry, to prove for each $j \in J$ that

$$\mathcal{A}_{\mathcal{P}} \vdash_{cl} x \in S_j?t_j + \sum_L x \in S_l?u_l = \sum_L x \in S_l?u_l.$$

Given this we can add each of these to obtain

$$\mathcal{A}_{\mathcal{P}} \vdash_{cl} \sum_J x \in S_j?t_j + \sum_L x \in S_l?u_l = \sum_L x \in S_l?u_l.$$

Now, this argument can be repeated, symmetrically, to yield

$$\mathcal{A}_{\mathcal{P}} \vdash_{cl} \sum_J x \in S_j?t_j + \sum_L x \in S_l?u_l = \sum_J x \in S_j?t_j$$

from which the result follows by TRANS.

So, choose an arbitrary j. For each $v \in S_j$ we know that $p \xrightarrow{v?} t_j[v/x]$. We know that $q \xrightarrow{v?} u_l[v/x]$ for some $l \in L$ such that $v \in S_l$ and $t_j[v/x] \sim_n u_l[v/x]$ because $p \simeq_n q$. Let $S_l^j = \{v \in S_j \cap S_l \mid u_l[v/x] \sim_n t_j[v/x]\}$. This gives a *finite* partition $\{S_l^j\}_{l \in L}$ of S_j such that $S_l^j \subseteq S_l$ for each $l \in L$. Then, by the idempotency of $+$ and the new axiom *Pattern* it is sufficient to show for each $l \in L$ that

$$\mathcal{A}_{\mathcal{P}} \vdash_{cl} x \in S_l^j?t_j + x \in S_l^j?u_l = x \in S_l?u_l.$$

This can be inferred from the rule cl-P-INPUT if we can prove for each $v \in S_l^j$

$$\mathcal{A}_{\mathcal{P}} \vdash_{cl} \tau!t_j[v/x] = \tau!u_l[v/x].$$

So let us fix a particular $v \in S_l^j$ and see how this can be inferred. We know that $v \in S_l$ and $t_j[v/x] \sim_n u_l[v/x]$. For convenience let p, q denote $t_j[v/x], u_l[v/x]$ respectively. We now apply Theorem 3.3.2 to get

$$p + x \in U?p \simeq_n q + x \in V?q$$

Where $U = I(q) - I(p)$ and $V = I(p) - I(q)$. We have four cases to consider.

1. $U = V = \emptyset$
   Since $x \in \emptyset?t \simeq_n \mathbf{O}$ we can immediately conclude that $p \simeq_n q$ and apply induction to obtain $\mathcal{A}_{\mathcal{P}} \vdash_{cl} p = q$ and therefore the required $\mathcal{A}_{\mathcal{P}} \vdash_{cl} \tau!p = \tau!q$.

2. $U = \emptyset, V \neq \emptyset$
   Here we have $p \simeq_n q + x \in V?q$ and again we can use induction to obtain $\mathcal{A}_{\mathcal{P}} \vdash_{cl} \tau!p = \tau!(q + x \in V?q)$. Now we can apply the *cl-P-Noisy* schema to obtain $\mathcal{A}_{\mathcal{P}} \vdash_{cl} \tau!q = \tau!(q + x \in V?q)$ from which the required result follows.

3. $U \neq \emptyset, V = \emptyset$
   Similar.

4. $U \neq \emptyset, V \neq \emptyset$
   Here we have $p + x \in U?p \simeq_n q + x \in V?q$ and in this case we can not apply induction immediately as the combined size of the terms has not decreased. But Thereom 3.3.2 tells us that there exists p', q' such that $d(p') < d(p)$ and $d(q') < d(q)$ such that $p' \sim_n p$ and $q' \sim_n q$. Suppose without loss of generality that $d(p) \leq d(q)$. Then, since $\tau!p \simeq_n \tau!p'$ we can use induction to obtain $\mathcal{A}_{\mathcal{P}} \vdash_{cl} \tau!p = \tau!p'$. Then a simple application of the cl-P-INPUT rule gives $\mathcal{A}_P \vdash_{cl} x \in U?p = x \in U?p'$. This in turn implies that $p + x \in U?p' \simeq_n q + x \in V?q$ and here we can apply induction since the combined size has decreased. So we obtain, as before, $\mathcal{A}_{\mathcal{P}} \vdash_{cl} \tau!(p + x \in U?p') = \tau!(q + x \in V?q)$. Using the fact that $\mathcal{A}_{\mathcal{P}} \vdash_{cl} x \in U?p = x \in U?p'$ we obtain $\mathcal{A}_{\mathcal{P}} \vdash_{cl} \tau!(p + x \in U?p) = \tau!(q + x \in V?q)$ from which the required $\mathcal{A}_{\mathcal{P}} \vdash_{cl} \tau!p = \tau!q$ follows by two applications of the *cl-P-Noisy* rule.

■

3.3.1 Pattern-matching with open terms

So far we have shown how to adapt the proof system of Section 3.2 based on closed terms to incorporate pattern-matching. Of course what we are really interested in is how to adapt the proof system based on open terms but, as a useful guide, we can follow the approach above to find the appropriate generalisation.

We first examined the axiom schema *cl-Noisy* to obtain a version of this suitable for $\mathcal{SPA}$, namely *cl-P-Noisy* . Recall that this schema used the construction $I(p)$, the set of values which a process can receive. In order to state *cl-P-Noisy* for open terms we would need to construct the set of values which a *term* can receive. Unfortunately this can't be defined as cleanly as $I(p)$. In fact, we need to talk about the set of values a term can receive relative to a boolean world. We will denote this by $I(b,t)$. An obvious property which we will require $I(b,t)$ to satisfy is

$$\delta \models b \text{ implies } I(b,t) = I(t\delta).$$

This $I(b,t)$ is not a trivial notion to characterise. For example consider the agent

$$t \equiv b_0 \gg x \in S_1?t' + \neg b_0 \gg x \in S_2?t''.$$

What would be the set of values which t can receive in the boolean world $\mathbf{tt}$, say? In any evaluation we know that either b_0 or $\neg b_0$ will be satisfied. So for some evaluations we may have $I(t\delta) = S_1$ and for others we may have $I(t\delta) = S_2$. If $I(b,t)$ is to satisfy the property stated above then it clearly makes no sense to ask what $I(\mathtt{tt},t)$ should be. The approach we take is to characterise the boolean expressions b for which $I(b,t)$ can have a meaningful definition which satisfies the required property.

The following function, *Bgs*, defined inductively over terms, will be useful in describing the booleans we are interested in.

- $Bgs(\mathbf{O}) = \emptyset$
- $Bgs(e!t) = \emptyset$
- $Bgs(x \in S?t) = \{(\mathtt{tt},S)\}$
- $Bgs(t+u) = Bgs(t) \cup Bgs(u)$
- $Bgs(b \gg t) = \{(b \wedge b',S) \mid (b',S) \in Bgs(t)\}$

We can then define

$$I(b,t) = \bigcup \{S \mid (b',S) \in Bgs(t), b \models b'\}.$$

Notice though that this definition, whilst intuitively appealing, may not accurately describe the set of values that are guaranteed to be received by t in the boolean world b. For instance, if we apply this definition of $I(\mathtt{tt},t)$ to the example above we see that we get the empty set of values. This is to be expected as we have already explained that we simply shouldn't be interested in $I(\mathtt{tt},t)$. The booleans b, for which $I(b,t)$ does yield a meaningful result are those booleans such that there is some subset K of $Bgs(t)$ with

$$b \models \bigwedge_{(b_k,S_k)\in K} b_k \wedge \bigwedge_{(b_j,S_j)\in Bgs(t)\setminus K} \neg b_j.$$

We call such booleans t-***uniform***.

We now show that this is a reasonable definition by relating $I(b,t)$ to $I(t\delta)$ where δ is an evaluation such that $\delta \models b$.

Lemma 3.3.4 *If b is t-uniform then*

$$\delta \models b \textit{ implies } I(t\delta) = I(b,t)$$

Proof b is t-uniform so there exists a set $K \subseteq Bgs(t)$ with the required property. We suppose that $\delta \models b$ and show by structural induction on t that $I(t\delta) = I(b,t)$.

The cases for **O**, and prefixing are trivial. We show the remaining two cases here.

Suppose that t is of the form $t_1 + t_2$. Then $I(t\delta) = I(t_1\delta) \cup I(t_2\delta)$. Using the sets $K_i = K \cap Bgs(t_i)$, for $i = 1,2$, we can see that b is both t_1 and t_2-uniform so by induction we get that $I(t\delta) = I(b,t_1) \cup I(b,t_2)$. But this is just $I(b,t_1+t_2)$.

Suppose that t is $b' \gg t_1$ and that $\delta \not\models b'$. We immediately have that $b \not\models b'$ which means $I((b' \gg t_1)\delta) = \emptyset = I(b, b' \gg t_1)$. We assume then that $\delta \models b'$. There are now two cases to consider: if K is empty then we know that

$$b \models \bigwedge_{(b_j,S_j)\in Bgs(t)} \neg b_j$$

and by definition, $I(b,t) = \emptyset$. We also know that $b \wedge b'$ must be t_1-uniform because each $(b_j,S_j) \in Bgs(b' \gg t)$ is of the form $b_j \equiv b' \wedge b''_j$ with $(b''_j,S_j) \in Bgs(t_1)$ so $b \models \neg(b' \wedge b''_j)$ for each j; in other words $b \wedge b' \models \neg b''_j$ for each j. Given this we can apply induction to obtain $I(t\delta) = I(t_1\delta) = I(b \wedge b', t_1)$. But this set is clearly empty. Hence $I(t\delta) = I(b,t) = \emptyset$.

So we must consider the case where K is non-empty. By uniformity we must have that $b \models b'$. This follows because each b_k in K is of the form $b' \wedge b''_k$ for some b''_k. In this case b must be t_1-uniform and induction gives

$$I(t\delta) = I(t_1\delta) = I(b,t_1).$$

But

$$\begin{array}{rcl} I(b,t_1) & = & \bigcup\{S \mid (b'',S) \in Bgs(t_1), b \models b''\} \\ & = & \bigcup\{S \mid (b' \wedge b'',S) \in Bgs(b' \gg t_1), b \models b' \wedge b''\} \\ & = & I(b,b' \gg t_1) \end{array}$$

■

We are now at a stage where we can state the axiom schema *cl-P-Noisy* for open terms.

$$\boxed{\textit{P-Noisy}: \quad b \rhd e!(t + x \in S?t) = e!t \quad \text{If } x \notin fv(t), b \text{ is } t\text{-uniform and } I(b,t) \cap S = \emptyset.}$$

The axioms *Pattern* and *Empty* can be used directly, without modification for open terms. However, we do need to attend to the rule for INPUT. The appropriate version of the input rule is:

$$\boxed{\text{P-INPUT:} \quad \frac{b \wedge x \in S \rhd \sum_I \tau!t_i = \sum_J \tau!u_j}{b \rhd \sum_I x \in S?t_i = \sum_J x \in S?u_j} \quad \text{if } x \notin fv(b).}$$

Note that, as before, we can derive a guarded version of this rule called P-INPUT$^{\gg}$ using Proposition 3.2.3.

Once again we write $\mathcal{A}_{\mathcal{P}} \vdash b \rhd t = u$ to mean that $b \rhd t = u$ can be derived from the axioms in $\mathcal{A}_{\mathcal{P}}$ ($\mathcal{A}$ plus *P-Noisy* , *Pattern* and *Empty*) using the proof system in Figure 3.3 with the modified input rule, P-INPUT.

Proposition 3.3.5 *(Soundness)*

$$\textit{If } \mathcal{A}_{\mathcal{P}} \vdash b \rhd t = u \textit{ and } \delta \models b \textit{ then } t\delta \simeq_n u\delta.$$

Proof We need only show that the modified rules/axioms are sound. The *Pattern* axiom and *Empty* axiom are evident.

For *P-Noisy* we know that b is t-uniform so by Lemma 3.3.4 $\delta \models b$ implies $I(t\delta) = I(b,t)$. Given this we only need to show that $t\delta + x \in S?t\delta \sim_n t\delta$ whenever $S \cap I(t\delta) = \emptyset$ for $\delta \models b$. This follows easily.

For the rule P-INPUT, suppose $\delta \models b$. We need to show that $\sum_I (x \in S?t_i)\delta \simeq_n \sum_J x \in S?u_j)\delta$. The only non-trivial move to match is of the form $(x \in S?t_i)\delta \xrightarrow{v?} q$. Here q must be of the form $t_i\delta[v/x]$ where $v \in S$. Since $x \notin fv(b)$ we have that $\delta[v/x] \models b \wedge x \in S$. So by assumption we have $\sum_I \tau!t_i\delta[v/x] \simeq_n \sum_J \tau!u_j\delta[v/x]$ which means $t_i\delta[v/x] \sim_n u_j\delta[v/x]$ for some j. Therefore $(\sum_J x \in S?u_j)\delta \xrightarrow{v?} u_j\delta[v/x]$ matches the move from p. ■

The remainder of this Section is the proof of completeness which, as before, requires heavy use of symbolic bisimulations. The addition of pattern sets to the syntax of the language necessitates enriching both the definition of the symbolic semantics of Figure 3.4, and of symbolic bisimulation. This takes us beyond the standard symbolic graphs described in Chapter 2. Firstly, we extend our abstract operational semantics to incorporate the pattern sets: see Figure 3.5. Again, symmetric rules for $+$ have been elided. Recalling the abbreviation $x?t$ for the term $x \in Val?t$ we see that

Discard	Input	Output
$\mathbf{O} \overset{\mathbf{tt},Val:}{\longmapsto} \mathbf{O}$		
$x \in S?t \overset{\mathbf{tt},Val\backslash S:}{\longmapsto} x \in S?t$	$\dfrac{}{x \in S?t \overset{\mathbf{tt},x\in S?}{\longmapsto} t}$	
$e!t \overset{\mathbf{tt},Val:}{\longmapsto} e!t$		$e!t \overset{\mathbf{tt},e!}{\longmapsto} t$
$\dfrac{t \overset{b,S:}{\longmapsto} t \quad u \overset{b',S':}{\longmapsto} u}{t+u \overset{b'\wedge b,S\cap S':}{\longmapsto} t+u}$	$\dfrac{t \overset{b,x\in S?}{\longmapsto} t'}{t+u \overset{b,x\in S?}{\longmapsto} t'}$	$\dfrac{t \overset{b,e!}{\longmapsto} t'}{t+u \overset{b,e!}{\longmapsto} t'}$
$b' \gg t \overset{\neg b',Val:}{\longmapsto} b' \gg t$		
$\dfrac{t \overset{b,S:}{\longmapsto} t}{b' \gg t \overset{b,S:}{\longmapsto} b' \gg t}$	$\dfrac{t \overset{b,x\in S?}{\longmapsto} t'}{b' \gg t \overset{b'\wedge b,x\in S?}{\longmapsto} t'}$	$\dfrac{t \overset{b,e!}{\longmapsto} t'}{b' \gg t \overset{b'\wedge b,e!}{\longmapsto} t'}$

Figure 3.5. Patterned abstract operational semantics

the extension is a conservative one. The transitions relations are, as before, labelled with boolean values acting as guards. The differences occur in transitions of the form $\overset{b,x\in S?}{\longmapsto}$ now decorated with the patterned input, and $\overset{b,S:}{\longmapsto}$ where S records the set of values which may be discarded. Our next move is to present the notion of a ***patterned noisy symbolic bisimulation*** which takes into account that a term demanding a matching transition for $t \overset{b,x\in S?}{\longmapsto} t'$, does so in a boolean world $b \wedge x \in S$. That is to say, the fact that x only ranges over values in S is accommodated in the condition which is partitioned.

Suppose $\mathbf{S} = \{S^b\}$ is a boolean indexed family of relations. Define $\mathcal{PNSB}(\mathbf{S})$ to be the family of relations such that

$(t,u) \in \mathcal{PNSB}(\mathbf{S})^b$ if whenever

- $t \overset{b_1,e!}{\longmapsto} t'$ there exists a $b \wedge b_1$-partition, B, and for each $b' \in B$ there exists a $u \overset{b_2,e'!}{\longmapsto} u'$ such that $b' \models b_2$, $b' \models e = e'$ and $(t',u') \in S^{b'}$

- $t \overset{b_1,x\in S?}{\longmapsto} t'$ there exists a variable z such that $z \notin fv(b,t,u)$ and a $b \wedge b_1 \wedge z \in S$-partition, B, and for each $b' \in B$ there exists $u \overset{b_2,y\in S'?}{\longmapsto} u'$ for some y and some S' such that $b' \models b_2$, $b' \models z \in S'$ and $(t'[z/x], u'[z/y]) \in S^{b'}$ *or* there exists $u \overset{b_2,S':}{\longmapsto} u$ such that $b' \models b_2$, $b' \models z \in S'$ and $(t'[z/x], u) \in S^{b'}$

(There are symmetric conditions for u also).

We call $\{S^b\}$ a patterned noisy symbolic bisimulation if $S^b \subseteq \mathcal{PNSB}(\mathbf{S})^b$ for each b and denote the largest such $\mathbf{S}$ by $\{\sim^b_{pn}\}$. Once again we now use the definition of $\sim^b_{pn}$ to define $\simeq^b_{pn}$, the largest congruence contained in $\sim^b_{pn}$:

$t \simeq^b_{pn} u$ if whenever

- $t \overset{b_1,e!}{\longmapsto} t'$ there exists a $b \wedge b_1$-partition, B, and for each $b' \in B$ there exists a $u \overset{b_2,e'!}{\longmapsto} u'$ such that $b' \models b_2$, $b' \models e = e'$ and $t' \sim^{b'}_{pn} u'$

- $t \overset{b_1,x\in S?}{\longmapsto} t'$ there exists a variable z such that $z \notin fv(b,t,u)$ and a $b \wedge b_1 \wedge z \in S$-partition, B, such that for each $b' \in B$ there exists a $u \overset{b_2,y\in S'?}{\longmapsto} u'$ such that $b' \models b_2$, $b' \models z \in S'$ and $t'[z/x] \sim_{pn}^{b'} u'[z/y]$

(Again symmetric conditions on u are omitted).

Once more, this version of symbolic bisimulation characterises the corresponding concrete version.

Proposition 3.3.6 *For any $t,u \in \mathcal{SPA}$*

$$t \simeq_{pn}^{b} u \text{ iff } (\forall\delta\cdot\delta \models b \text{ implies } t\delta \simeq_n u\delta).$$

Proof We show that the proposition holds for $\sim_{pn}^{b}$ as the result follows easily from this. The *only if* part is straightforward: given a noisy symbolic bisimulation $\{S^b\}$, we define the relation $R = \{(t\delta,u\delta) \mid \exists b\cdot\delta \models b \text{ and } (t,u) \in S^b\}$ and show that this is a noisy bisimulation.

For the *if* part we suppose that we have a noisy bisimulation R and define

$$S^b = \{(t,u) \mid \delta \models b \text{ implies } (t\delta,u\delta) \in R\}.$$

We aim to show that $S^b \subseteq \mathcal{PNSB}(\mathbf{S})^b$. Suppose that $(t,u) \in S^b$ and that $t \overset{b_1,x\in S?}{\longmapsto} t'$. Let

$$\mathcal{U} = \left\{u' \mid u \overset{b(u'),\alpha(u')}{\longmapsto} u', \alpha(u') \in \{x \in S'?, S':\}\right\}$$

and let $u' \in \mathcal{U}$. Define $b_{u'}^{aux}$ such that $\delta \models b_{u'}^{aux}$ iff $(t'\delta,u'\delta) \in R$ and define

$$b_{u'} = b(u') \wedge x \in S' \wedge b_{u'}^{aux}$$

where S' is the set given by the action $\alpha(u')$. We have defined $b_{u'}^{aux}$ abstractly here and we simply assume that such a boolean exists in our property metalanguage. This is the same expressivity requirement used in the proof of Theorem 3.2.7. We let B be the set $\{b \wedge b_1 \wedge x \in S \wedge b_{u'} \mid u' \in \mathcal{U}\}$. It is clear that $\bigvee B \models b \wedge b_1 \wedge x \in S$ so we show the converse. Suppose $\delta \models b \wedge b_1 \wedge x \in S$. Then $\delta = \delta[v_0/x]$ for some $v_0 \in S$ and $(t\delta,u\delta) \in R$. By a simple generalisation of Proposition 3.2.6 we know that $t\delta \xrightarrow{v?} t'\delta[v/x]$, for any $v \in S$. So we know that there exists a matching move $u\delta \xrightarrow{v?} u_v\delta[v/x]$ or $u\delta \xrightarrow{v:} u_v\delta[v/x]$ such that $(t'\delta[v/x],u_v\delta[v/x]) \in R$.

By the same generalised proposition we see there exists a $u_v \in \mathcal{U}$ such that $u \overset{b(u_v),\alpha(u_v)}{\longmapsto} u_v$ with $\delta \models b(u_v)$ and $v \in S'$ (S' being the set of $\alpha(u_v)$). In particular we have this for v_0 so $(t'\delta,u_{v_0}\delta) \in R$. Which implies that $\delta \models b_{u_{v_0}}$. Therefore $\delta \models \bigvee B$.

This gives us the partition required. For each $b' = b \wedge b_1 \wedge x \in S \wedge b_{u'} \in B$ we have $u \overset{b(u'),\alpha(u')}{\longmapsto} u'$ with $b' \models b(u')$, $b' \models x \in S'$ and $b' \models b_{u'}^{aux}$. This implies that $(t'\delta,u'\delta) \in R$, hence $(t',u') \in S^{b'}$.

The symbolic transmissions from t can be dealt with in a similar, slightly simpler, manner. ■

Having developed a suitable notion of patterned symbolic bisimulation we develop a version of Theorem 3.3.2 for open terms.

Theorem 3.3.7 *If t and u are standard forms then $t \sim_{pn}^{b} u$ if and only if there exists a b-partition, B, such that each $b' \in B$ is both t and u-uniform and*

$$t + z \in S?t \;\simeq_{pn}^{b'}\; u + z \in S'?u$$

where $z \notin fv(b,t,u), S = (I(b',u) - I(b',t))$ and $S' = (I(b',t) - I(b',u))$. Moreover, when both S and S' are non-empty, there exist t',u' such that $d(t') < d(t), d(u') < d(u)$ and $t' \sim_{pn}^{b'} t, u' \sim_{pn}^{b'} u$.

Proof The *if* direction is easy so we show the converse. We use Lemma 3.3.4 We exhibit two b-partitions, B_1, B_2 such that each $b_1 \in B_1$ is t-uniform and each $b_2 \in B_2$ is u-uniform. Let

$$t \equiv \sum_I c_i \gg \alpha_i.t_i \text{ and } u \equiv \sum_J d_j \gg \beta_j.u_j$$

be the standard forms and, given $K \subseteq I_?$ we define

$$b_K = \bigwedge_{k \in K} c_k \wedge \bigwedge_{i \in I_? \setminus K} \neg c_i$$

(similarly for $L \subseteq J$). The two partitions are simply $B_1 = \{b \wedge b_K \mid K \subseteq I_?\}$ and $B_2 = \{b \wedge b_L \mid L \subseteq J_?\}$. We let $B = \{b_1 \wedge b_2 \mid b_1 \in B_1, b_2 \in B_2\}$ and it is trivial to see that each $b' \in B$ is in fact t and u-uniform and, by Lemma 3.3.4 B has the property that for each $b' \in B$, $\delta \models b'$ implies $I(t\delta) = I(b',t)$ and $I(u\delta) = I(b',u)$. Suppose $\delta \models b'$. Then $t\delta \sim_n u\delta$. We apply Theorem 3.3.2 to get

$$t\delta + z \in S?t\delta \simeq_n u\delta + z \in S'?u\delta$$

where $S = I(u\delta) - I(t\delta)$ and $S' = I(t\delta) - I(u\delta)$. By properties of the partition we know that $S = I(b',u) - I(b',t)$ and $S' = I(b',t) - I(b',u)$. This is true for each $\delta \models b'$ so it follows from Proposition 3.3.6 that $t + z \in S?t \simeq_{pn}^{b'} u + z \in S'?u$.

What remains to be proved is the existence of t', u', in the case where S, S', are non-empty. This also follows from Theorem 3.3.2, save for a mild complication. For each $\delta \models b'$ there is a p', q' such that $p' \sim_n t\delta$ and $q' \sim_n u\delta$. These p', q' are obtained by considering two values $v_1 \in S, v_2 \in S'$ and using the fact that $t\delta \xrightarrow{v_2?} p'$ and $u\delta \xrightarrow{v_1?} q'$. This means that p' is of the form $(t_i[v_2/x])\delta$ and similarly q' is of the form $(u_j[v_1/x])\delta$. Let t'_δ denote this $t_i[v_2/x]$ and let u'_δ denote $u_j[v_1/x]$. For different δ these t'_δ, u'_δ may be different terms although there are only finitely many of them to choose from.

To resolve this we partition b' further. Define

$$\Sigma_{ij} = \{\delta \mid \delta \models b' \text{ and } t'_\delta \equiv t_i[v_2/x] \text{ and } u'_\delta \equiv u_j[v_1/x]\}$$

and define $\delta \models b_{ij}$ if and only if $\delta \in \Sigma_{ij}$. These $\{b_{ij}\}$ partition b' and for each $\delta \models b_{ij}$ we have terms t', u', independent of δ, such that $t'\delta \sim_n t\delta$ and $u'\delta \sim_n u\delta$. ■

Theorem 3.3.8 *(Completeness)* $t \simeq_{pn}^{b} u$ *implies* $\mathcal{A}_{\mathcal{P}} \vdash b \rhd t = u$.

Proof As before we adopt standard forms for t, u and proceed by induction on the sum of the depths, $d(t) + d(u)$. Again it is sufficient to show

$$\mathcal{A}_{\mathcal{P}} \vdash b \rhd t_! = t_! \text{ and } \mathcal{A}_{\mathcal{P}} \vdash b \rhd t_? = u_?$$

independently. Suppose

$$t_? \equiv \sum_{I_?} c_i \gg z \in S_i?t_i$$

where $z \notin fv(b,t,u)$. We wish to assume the property that whenever $t \overset{c_i, z \in S_i?}{\longmapsto} t_i$ we get a matching partition such that each element is of the form $b' \wedge z \in S_i$ where the b' form a $b \wedge c_i$-partition. We can grant this assumption if we transform the standard forms slightly, as follows.

We have (for each i) that $t \overset{c_i, z \in S_i?}{\longmapsto} t_i$. Since $t \simeq_{pn}^{b} u$ we know that there exists a matching $b \wedge c_i \wedge z \in S_i$-partition, B. Because $z \notin fv(b,c_i)$ we know that each element of B is logically equivalent to something of the form $b' \wedge z \in S_{i_k}$ (for some indexing set K) where $\bigvee b' \equiv b \wedge c_i$ and $\bigcup S_{i_k} = S_i$. We use the axiom *Pattern* to decompose the summand $x_i \in S_i?t_i$ of t into the sum $\sum_{k \in K} x_i \in S_{i_k}?t_i$ and Proposition 3.2.3 to distribute c_i across this sum. We repeat this for each $i \in I_?$ and also for u.

Having done this we show $\mathcal{A}_{\mathcal{P}} \vdash b \rhd t_? = u_?$ where $t_? \equiv \sum_I c_i \gg z \in S_i?t_i$ and $u_? \equiv \sum_J d_j \gg z \in S_j?u_j$ are now the modified standard forms. It is sufficient to show

$$\mathcal{A}_{\mathcal{P}} \vdash b \wedge c_i \rhd u_? = u_? + c_i \gg z \in S_i?t_i$$

for each $i \in I?$. For once we have obtained this we can apply GUARD and add to get

$$\mathcal{A}_{\mathcal{P}} \vdash b \rhd u_? = u_? + t_?.$$

Repeating this argument we obtain a symmetric version of this which results in $\mathcal{A}_{\mathcal{P}} \vdash b \rhd t_? = u_?$ by TRANS.

So for an arbitrary $i \in I_?$ we now show

$$\mathcal{A}_{\mathcal{P}} \vdash b \wedge c_i \rhd u_? = u_? + c_i \gg z \in S_i?t_i.$$

Suppose that $t \overset{c_i,z\in S_i?}{\longmapsto} t_i$. Since $t \simeq^b_{pn} u$ we know there exists a $b \wedge c_i \wedge z \in S_i$-partition, B_i, with the property that each element of B_i is of the form $b' \wedge z \in S_i$ (where the b' partition $b \wedge c_i$) and, for each such b', there exists $u \overset{d_j,z\in S_j?}{\longmapsto} u_j$ such that $b' \wedge z \in S_i \models d_j$, $b' \wedge z \in S_i \models z \in S_j$ and $t_i \sim^{b' \wedge z \in S_i}_{pn} u_j$. Let $J_i \subseteq J_?$ denote the set of j indices used in this particular partition. The fact that $b' \wedge z \in S_i \models z \in S_j$ gives us that $S_i \subseteq S_j$ for each $j \in J_i$.

The approach we take is to prove that

$$\mathcal{A}_{\mathcal{P}} \vdash b' \rhd c_i \gg z \in S_i?t_i + \sum_{j \in J_i} d_j \gg z \in S_i?u_j = \sum_{j \in J_i} d_j \gg z \in S_i?u_j$$

for each $b' \in B$. We can then use the CASE rule to obtain

$$\mathcal{A}_{\mathcal{P}} \vdash b \wedge c_i \rhd c_i \gg z \in S_i?t_i + \sum_{j \in J_i} d_j \gg z \in S_i?u_j = \sum_{j \in J_i} d_j \gg z \in S_i?u_j.$$

If we now add $u_?$ to each side, we can use idempotence of $+$ and axiom *Pattern* (because $S_i \subseteq S_j$ for each $j \in J_?$) to get

$$\mathcal{A}_{\mathcal{P}} \vdash b \wedge c_i \rhd c_i \gg z \in S_i?t_i + u_? = u_?$$

as required.

We know that for each b' we have $t_i \sim^{b' \wedge z \in S_i}_{pn} u_j$ for some $j \in J_i$. Assuming standard forms for t_i and u_j we appeal to Theorem 3.3.7 for a $b' \wedge z \in S_i$-partition, B', such that for each $b'' \in B'$ we have that b'' is both t_i and u_j-uniform and that

$$t_i + x \in S?t_i \simeq^{b''}_{pn} u_j + x \in S'?u_j$$

where S is $I(b'', u_j) - I(b'', t_i)$ and S' is $I(b'', t_i) - I(b'', u_j)$. Once again we have four cases to consider. In each of which we demonstrate

$$\mathcal{A}_{\mathcal{P}} \vdash b'' \rhd \tau!t_i = \tau!u_j.$$

1. $S = S' = \emptyset$.
   Since $x \in \emptyset?t \simeq^{\mathbf{tt}}_{pn} \mathbf{O}$ we conclude that $t_i \simeq^{b''}_{pn} u_j$ and apply induction to obtain $\mathcal{A}_{\mathcal{P}} \vdash b'' \rhd t_i = u_j$. Whence $\mathcal{A}_{\mathcal{P}} \vdash b'' \rhd \tau!t_i = \tau!u_j$.

2. $S = \emptyset, S' \neq \emptyset$.
   We have $t_i \simeq^{b''}_{pn} u_j + x \in S'?u_j$. Again we apply induction and use TAU and axiom *P-Noisy* to obtain $\mathcal{A}_{\mathcal{P}} \vdash b'' \rhd \tau!t_i = \tau!u_j$.

3. $S \neq \emptyset, S' = \emptyset$.
   Similar.

4. $S \neq \emptyset, S' \neq \emptyset$.
   By Theorem 3.3.7 there exists t', u' such that $t_i \sim_{pn}^{b''} t'$ and $u_j \sim_{pn}^{b''} u'$ and $d(t') < d(t_i), d(u') < d(u_j)$. Without loss of generality, suppose that $d(t) \leq d(u)$. By induction we get $\mathcal{A}_{\mathcal{P}} \vdash b'' \rhd \tau!t_i = \tau!t'$. Whence $\mathcal{A}_{\mathcal{P}} \vdash b'' \rhd x \in S?t_i = x \in S?t'$ by P-INPUT. We can deduce that

   $$t_i + x \in S?t' \simeq_{pn}^{b''} u_j + x \in S'?u_j$$

   and see that induction is applicable here also. Therefore we get

   $$\mathcal{A}_{\mathcal{P}} \vdash b'' \rhd t_i + x \in S?t' = u_j + x \in S'?u_j.$$

   Using the previous result we can substitute t' for t_i then apply TAU and axiom *P-Noisy* twice to get $\mathcal{A}_{\mathcal{P}} \vdash b'' \rhd \tau!t_i = \tau!u_j$ as required.

So we have seen that for each $b'' \in B'$ we can prove $\mathcal{A}_{\mathcal{P}} \vdash b'' \rhd \tau!t_i = \tau!u_j$ so we apply CASE to give $\mathcal{A}_{\mathcal{P}} \vdash b' \wedge z \in S_i \rhd \tau!t_i = \tau!u_j$. We know that $b' \models c_i \wedge d_j$ and we can use Proposition 3.2.3 to give

$$\mathcal{A}_{\mathcal{P}} \vdash b' \wedge z \in S_i \rhd c_i \gg \tau!t_i + d_j \gg \tau!u_j = d_j \gg \tau!u_j.$$

We add in the rest of the J_i to get

$$\mathcal{A}_{\mathcal{P}} \vdash b' \wedge z \in S_i \rhd c_i \gg \tau!t_i + \sum_{j \in J_i} d_j \gg \tau!u_j = \sum_{j \in J_i} d_j \gg \tau!u_j$$

followed by an appllication of the P-INPUT$^{\gg}$ rule which yields the result

$$\mathcal{A}_{\mathcal{P}} \vdash b' \rhd c_i \gg z \in S_i?t_i + \sum_{j \in J_i} d_j \gg z \in S_i?u_j = \sum_{j \in J_i} d_j \gg z \in S_i?u_j.$$

■

3.4 Finite CBS

In the previous sections we have given a number of proof systems for the class of simple agents, $\mathcal{SPA}$. In order to extend these to full finite CBS it remains only to to consider the parallel and translation constructs. The standard approach is to introduce axioms or axiom schemas which are sufficient to translate agents of finite CBS into agents of $\mathcal{SPA}$. The parallel operator is usually treated using an expansion theorem while translations, being generalisations of the restriction and renaming operators of CCS, can be handled by a set of axiom schemas which when used as rewrite rules can reduce a term of the form $p_{(f,g)}$, where $p \in \mathcal{SPA}$, to a term in $\mathcal{SPA}$. We give an outline of the necessary axiom schemas.

The expansion theorem presented in [86] is not sufficient here as we need to deal with the pattern sets on inputs. Moreover, the expansion is complicated by the fact that we are working over open terms. However a suitable version of expansion is presented in Figure 3.6:

Proposition 3.4.1 *Suppose $x \notin fv(t,u)$ and*

$$t \equiv \sum_{i \in I_!} c_i \gg e_i!t_i + \sum_{i \in I_?} c_i \gg x \in S_i?t_i$$

and

$$u \equiv \sum_{j \in J_!} d_j \gg e_j!u_j + \sum_{j \in J_?} d_j \gg x \in S_j?u_j.$$

Then $t|u \simeq_n Exp(t|u)$.

Writing $\bar{S}_K \overset{def}{=} \bigcap_{k\in K} (Val - S_k)$, let

$$\begin{array}{rcll}
Exp(t \mid u) & = & \sum_{i\in I_!,\, j\in J_?} & (c_i \wedge d_j \wedge e_i \in S_j) \gg e_i!(t_i \mid u_j[e_i/x]) \\
 & + & \sum_{i\in I_?,\, j\in J_!} & (c_i \wedge d_j \wedge e_j \in S_i) \gg e_j!(t_i[e_j/x] \mid u_j) \\
 & + & \sum_{i\in I_!, K\subseteq J_?} & (c_i \wedge \bigwedge_{k\in K} \neg d_k \wedge e_i \in \bar{S}_{J_?-K}) \gg e_i!(t_i \mid u) \\
 & + & \sum_{j\in J_!, K\subseteq I_?} & (\bigwedge_{k\in K} \neg c_k \wedge d_j \wedge e_j \in \bar{S}_{I_?-K}) \gg e_j!(t \mid u_j) \\
 & + & \sum_{i\in I_?,\, j\in J_?} & (c_i \wedge d_j) \gg x \in S_i \cap S_j ?(t_i \mid u_j) \\
 & + & \sum_{i\in I_?, K\subseteq J_?} & (c_i \wedge \bigwedge_{k\in K} \neg d_k) \gg x \in (S_i \cap \bar{S}_{J_?-K})?(t_i \mid u) \\
 & + & \sum_{j\in J_?, K\subseteq I_?} & (\bigwedge_{k\in K} \neg c_k \wedge d_j) \gg x \in (S_j \cap \bar{S}_{I_?-K})?(t \mid u_j).
\end{array}$$

Figure 3.6. Expansion laws for CBS parallel

Proof This can easily be proved directly from the operational semantics. ■

To accommodate the translation functions we use the following coding defined inductively on terms. This coding requires that we extend the signature of the data-domain with the function symbols used to express the f and g translation functions. Let $g^{-1}(S) = \{v \in Val \mid g(v) \in S\}$. Note that $\tau \notin g^{-1}(S)$ as $\tau \notin S$ and g is strict. We use Λ to denote a function from Var to translation functions and we let $e\Lambda$ denote the substitution $e[g(x)/x \mid x \in fv(e), g = \Lambda(x)]$.

- $\langle \mathbf{O} \rangle_{(f,g,\Lambda)} = \mathbf{O}$
- $\langle e!t \rangle_{(f,g,\Lambda)} = f(e\Lambda)!\langle t \rangle_{(f,g,\Lambda)}$
- $\langle x \in S?t \rangle_{(f,g,\Lambda)} = x \in g^{-1}(S)?\langle t \rangle_{(f,g,\Lambda[g/x])}$
- $\langle b \gg t \rangle_{(f,g,\Lambda)} = b\Lambda \gg \langle t \rangle_{(f,g,\Lambda)}$
- $\langle \sum_{i\in I} t_i \rangle_{(f,g,\Lambda)} = \sum_{i\in I} \langle t_i \rangle_{(f,g,\Lambda)}$
- $\langle t_{(f',g')} \rangle_{(f,g,\Lambda)} = \langle t \rangle_{(f\cdot f', g'\cdot g, \Lambda)}$

The idea here is to ensure that any broacast transmission from $\langle t \rangle_{(f,g,\Lambda)}$ is translated using the function f. When a process $p_{(f,g)}$ appears to receive a value v we expect the continuing process to be something of the form $p'[v/x]$ but we see by the operational semantics, Figure 3.1, that the process $p_{(f,g)}$, whilst appearing to receive v, actually receives the value $g(v)$ and continues to behave like $p'[g(v)/x]$. To capture this behaviour in the coding we need to record, using the Λ function, exactly which translation g was used when x became unbound and then subsequently use that translation wherever x occurs.

Lemma 3.4.2 *If* $\Lambda(x) = Id$ *then* $\langle t \rangle_{(f,g,\Lambda[h/x])}\delta[v/x] \equiv \langle t \rangle_{(f,g,\Lambda)}\delta[h(v)/x]$.

Proof Structural induction on t. the interesting cases are when t is a prefixed term.

Suppose t is $e!t'$. then

$$\langle t \rangle_{(f,g,\Lambda[h/x])}\delta[v/x] \equiv f(e\Lambda[h/x]\delta[v/x])!\langle t' \rangle_{(f,g,\Lambda[h/x])}\delta[v/x].$$

Which, by induction, is equivalent to

$$f(e\Lambda\delta[h(v)/x])!\langle t'\rangle_{(f,g,\Lambda)}\delta[h(v)/x].$$

But this is just $\langle t\rangle_{(f,g,\Lambda)}\delta[h(v)/x]$.

Suppose t is $y \in S?t'$ and suppose without loss of generality that $x \neq y$. We have

$$\langle t\rangle_{(f,g,\Lambda[h/x])}\delta[v/x] \equiv y \in g^{-1}(S)?\langle t'\rangle_{(f,g,\Lambda[h/x][g/y])}\delta[v/x].$$

We can write $\Lambda[h/x][g/y]$ as $\Lambda[g/y][h/x]$ and apply induction to see that

$$\langle t\rangle_{(f,g,\Lambda[h/x])}\delta[v/x] \equiv y \in g^{-1}(S)?\langle t'\rangle_{(f,g,\Lambda[g/y])}\delta[h(v)/x]$$

which is $\langle t\rangle_{(f,g,\Lambda)}\delta[h(v)/x]$. ■

Proposition 3.4.3 *If we write* 1 *for the* Λ *function which is constantly the identity translation then* $\langle t\rangle_{(f,g,1)} \simeq_n t_{(f,g)}$.

Proof Structural induction on t. Again we show the interesting cases where t is a prefixed term.

Suppose t is $e!t'$, then $\langle t\rangle_{(f,g,1)} = f(e)!\langle t'\rangle_{(f,g,1)}$. Induction tells us that this is $f(e)!t'_{(f,g)}$ which is easily seen to be $\simeq_n$-equivalent to $t_{(f,g)}$.

Suppose t is $x \in S?t'$. We need to show that for any δ we have a matching transition for

$$x \in g^{-1}(S)?\langle t'\rangle_{(f,g,[g/x])}\delta \xrightarrow{v?} \langle t'\rangle_{(f,g,[g/x])}\delta[v/x]$$

for all v such that $g(v) \in S$. Clearly, if $g(v) \in S$ then $t_{(f,g)} \xrightarrow{v?} t'_{(f,g)}\delta[g(v)/x]$. We know by induction that $t'_{(f,g)}\delta[g(v)/x] \simeq_n \langle t'\rangle_{(f,g,1)}\delta[g(v)/x]$ and by the previous Lemma that this is $\langle t'\rangle_{(f,g,[g/x])}\delta[v/x]$. A similar argument applies for finding matches for transitions from $t_{(f,g)}$. ■

The identity in Proposition 3.4.1 can be viewed as an axiom schema, which we call EXP, while TRANS is used to denote the obvious axiom schemas underlying the above encoding; each line gives rise to a separate axiom schema.

Theorem 3.4.4 *For any two terms of finite CBS, t and u,*

$$t \sim^b_{pn} u \text{ iff } \mathcal{A}_{\mathcal{P}}, EXP, TRANS \vdash b \rhd t = u.$$

Proof Use soundness of translations along with soundness and completeness results of the previous section. ■

By specialising the above Theorem a similar result can be obtained for the proof system for the class of closed terms of CBS.

Chapter 4

Weak Bisimulation for a Calculus of Broadcasting Systems

In the previous chapter we addressed the question of finding a suitable, bisimulation based, semantic equivalence for the language CBS [86]. The equivalence we studied was derived under the assumption that the production of noise $\tau!$ in the language was treated no differently than the production of any other value. The present chapter seeks to investigate this same equivalence but this time treating noise as unobservable. This gives us the semantic notion of *weak* bisimulation equivalence. As before we define our equivalence by considering agents' possible broadcast actions within contexts, *i.e.* weak barbed bisimulation. Symbolic techniques, similar to those of Chapter 3, are used to show how a complete axiomatisation of this weakened bisimulation may be obtained.

4.1 Weak barbed bisimulation

Recall that the definition of barbed bisimulation simply required a notion of reduction $p \xrightarrow{\tau!} p'$ and a notion of barb $p \downarrow v$. The corresponding weak version of this equivalence also requires these two elements in a weaker form. We define ***weak transitions***, traditionally denoted by a double arrow, as the least relations between agents of CBS such that

- $p \overset{\varepsilon}{\Longrightarrow} p$
- $p \xrightarrow{\alpha} q$ implies $p \overset{\alpha}{\Longrightarrow} q$
- $p \xrightarrow{\tau!} \overset{\alpha}{\Longrightarrow} q$ implies $p \overset{\alpha}{\Longrightarrow} q$
- $p \overset{\alpha}{\Longrightarrow} \xrightarrow{\tau!} q$ implies $p \overset{\alpha}{\Longrightarrow} q$

where $\alpha \in \{w!, v?, w:\}$ and ε is a dummy symbol. We will occasionally use the notation $P \overset{\tau!\alpha}{\Longrightarrow} Q$ to mean $P \overset{\tau!}{\Longrightarrow} \overset{\alpha}{\Longrightarrow} Q$, and we will define $\hat{\alpha}$ to be ε when $\alpha = \tau!$ and α otherwise.

We turn now to the definition of a *weak* semantic equivalence, which abstracts away from the occurrence of $\tau!$ actions. For any value v let $p \Downarrow v$ mean that there exists a p' such that $p \xrightarrow{v!} p'$.

A relation $\mathcal{R}$ between agents is called a ***weak barbed bisimulation*** if for each $(p,q) \in \mathcal{R}$ we have

- whenever $p \xrightarrow{\tau!} p'$ then $q \overset{\varepsilon}{\Longrightarrow} q'$ for some q' such that $(p',q') \in \mathcal{R}$
- whenever $q \xrightarrow{\tau!} q'$ then $p \overset{\varepsilon}{\Longrightarrow} p'$ for some p' such that $(p',q') \in \mathcal{R}$

- if $p \downarrow v$ then $q \overset{\varepsilon}{\Longrightarrow} q'$ for some q' such that $q' \downarrow v$
- if $q \downarrow v$ then $p \overset{\varepsilon}{\Longrightarrow} p'$ for some p' such that $p' \downarrow v$.

We write $p \approx_{barb} q$ if there exists a weak barbed bisimulation containing (p,q).

Of course $\approx_{barb}$ is uninteresting in itself and we are really concerned with the congruence associated with this relation.

For agents p and q let $p \cong_{barb} q$ if $C[p] \approx_{barb} C[q]$ for every CBS context $C[\_]$.

Again we will write $(p \cong_{barb} q) : V$ to specify that the contexts used are built over the value set V. The intention is to characterise $\cong_{barb}$ as a weakened version of noisy bisimulation. In order to express noisy bisimulation neatly we introduced the idea of a reaction relation $p \xrightarrow{v??} q$. Similarly we will write

$$p \overset{v??}{\Longrightarrow} q \text{ iff } p \overset{v?}{\Longrightarrow} q \text{ or } p \overset{v:}{\Longrightarrow} q.$$

The characterisation is as follows: A relation $\mathcal{R}$ is a ***weak bisimulation*** if for each $(p,q) \in \mathcal{R}$ we have

- whenever $p \xrightarrow{w!} p'$ then $q \overset{\hat{w}!}{\Longrightarrow} q'$ for some q' such that $(p',q') \in \mathcal{R}$.
- whenever $p \xrightarrow{v??} p'$ then $q \overset{v??}{\Longrightarrow} q'$ for some q' such that $(p',q') \in \mathcal{R}$
- whenever $q \xrightarrow{w!} q'$ then $p \overset{\hat{w}!}{\Longrightarrow} p'$ for some p' such that $(p',q') \in \mathcal{R}$.
- whenever $q \xrightarrow{v??} q'$ then $p \overset{v??}{\Longrightarrow} p'$ for some p' such that $(p',q') \in \mathcal{R}$.

We write $p \approx q$ if there exists a weak bisimulation $\mathcal{R}$ such that $(p,q) \in \mathcal{R}$.

This is the definition of weak bisimulation proposed in [86], and here we justify the choice using $\cong_{barb}$.

In Proposition 3.1.2 we proved that discard need not be taken into account when defining noisy bisimulation. Unfortunately the same is not true for weak bisimulation in CBS. To illustrate this suppose that $\approx'$ is the largest of the relations $\mathcal{R}$ such that for each $(p,q) \in \mathcal{R}$ we have

- whenever $p \xrightarrow{w!} p'$ then $q \overset{\hat{w}!}{\Longrightarrow} q'$ for some q' such that $(p',q') \in \mathcal{R}$.
- whenever $p \xrightarrow{v?} p'$ then $q \overset{v??}{\Longrightarrow} q'$ for some q' such that $(p',q') \in \mathcal{R}$

(with symmetric conditions on q). Then it is easy to see that $\tau!p \approx' p$ for any agent p although $\tau!p$ is not, in general, weakly bisimilar to p. A counter-example to illustrate this is

$$\tau!x \in Val?x!\mathbf{O} \not\approx x \in Val?x!\mathbf{O}.$$

This is because the agent $\tau!x?x!\mathbf{O} \xrightarrow{v:} \tau!x?x!\mathbf{O}$ for any value $v \in Val$. In order to match this move the agent $x \in Val?x!\mathbf{O}$ must perform a reception, *i.e.* $x \in Val?x!\mathbf{O} \xrightarrow{v?} v!\mathbf{O}$ and the resulting processes are not weakly bisimilar. The counter-example also serves to show that $\approx'$ is not preserved by parallel composition by using the context $v!v'!\mathbf{O}|[\_]$ for $v \neq v'$. In fact the relation obtained by closing $\approx'$ under parallel composition coincides with $\approx$.

Proposition 4.1.1

$$p \approx q \;\; \textit{iff} \;\; (p|r) \approx' (q|r) \;\; \textit{for all } r : Val^+.$$

Proof It is simple to show that $\approx$ is preserved by parallel, $|$. So we see that $p \approx q$ implies $p|r \approx q|r$ for any r, which means that $p|r \approx' q|r$ because $\approx \subset \approx'$.

The converse requires the construction of a particular agent r_0 which will serve to show $p \approx q$. Let

$$r_0 \Longleftarrow \sum_{v \in Val} v!(v'!\mathbf{O} + \tau!r_0)$$

where the $v' \in Val^+$ cannot appear in p, q. A simple argument now shows that

$$\mathcal{R} = \{(p,q) \mid (p|r_0) \approx' (q|r_0)\}$$

is a weak bisimulation. The non-trivial part of the proof is how to match a move $p \xrightarrow{v:} p$, for $(p,q) \in \mathcal{R}$. We see that $p|r_0 \xrightarrow{v!} p'$ where $p' \equiv p|(v'!\mathbf{O} + \tau!r_0)$. The condition $p|r_0 \approx' q|r_0$ guarantees us a move $q|r_0 \overset{v!}{\Longrightarrow} r'$ such that $p' \approx' r'$. We can discover that r' must be of the form $q'|(v'!\mathbf{O} + \tau!r_0)$, with $q \overset{v??}{\Longrightarrow} q'$, by considering the barb $p' \downarrow v'$. The move $p' \xrightarrow{\tau!} p|r_0$ will ensure that $r' \overset{\varepsilon}{\Longrightarrow} (q''|r_0)$ where $q' \overset{\varepsilon}{\Longrightarrow} q''$ and $p|r_0 \approx' q''|r_0$ so that $(p, q'') \in \mathcal{R}$. ■

Proposition 4.1.2 $\approx$ *is preserved by all of the CBS operators except summation.*

Proof Standard, see [74]. ■

We now show that it is possible to obtain this definition of weak bisimulation by considering barbed bisimulations in static contexts, that is contexts in which the hole does not appear as a summand in a choice. The contexts will be built over an extended value set Val^{++} which is defined to be the disjoint union of the sets Val, $Val' \overset{def}{=} \{v' \mid v \in Val\}, \{in, out, c\}$ and $\mathcal{N}$; we assume a successor function on $\mathcal{N}$ also.

Proposition 4.1.3 *If* $C[p] \approx_{barb} C[q]$ *for every static context* $C[\_]$ *built over* Val^{++} *then* $p \approx q$.

Proof We only outline the proof here as the details are similar to those in [94]. We define translation functions on $Val^{++} \cup \{\tau\}$ as follows:

$$g(w) = \begin{cases} w' & \text{if } w \in Val \\ w & \text{otherwise.} \end{cases}$$

$$f(w) = \begin{cases} \tau & \text{if } w \in Val \cup \{c\} \\ w & \text{otherwise.} \end{cases}$$

$$h(w) = \begin{cases} \tau & \text{if } w = c \\ w & \text{otherwise.} \end{cases}$$

We recall that the notation $g(x)!\mathbf{O}$ and $g(v)!\mathbf{O}$ is shorthand for $(x!\mathbf{O})_{(g,Id)}$ and $(v!\mathbf{O})_{(g,Id)}$ respectively. Armed with these translation functions we can build a collection of static contexts $C_n[\_]$ similar to those in [94]. A full explanation of the construction of Sangiorgi's contexts can be found in his thesis. Ours differ only in that we explicitly translate communicated values into τ actions using the translation functions and we require the use of a distinguished value c which plays the rôle of a private channel for communicating with and incrementing the counter.

We let $+$ denote binary choice and define

$$Count \Longleftarrow \lambda n.n!.\mathbf{O} + x \in \{c\}?Count(n+1)$$

and the constant D

$$\begin{aligned} D \quad \Longleftarrow \quad & x \in Val?c!c!(\tau!(g(x)!\mathbf{O} + out!\mathbf{O}) + \tau!D) \\ + \quad & \sum_{v \in Val} v!c!c!(\tau!(g(v)!\mathbf{O} + in!\mathbf{O}) + \tau!D) \\ + \quad & \tau!0!\mathbf{O} \\ + \quad & \tau!1!\mathbf{O} \end{aligned}$$

The contexts we require are defined by

$$C_n[\,\_\,] \Longleftarrow ([\,\_\,]_{(Id,h)}|D|Count(n))_{(f,Id)}.$$

Given these we define a relation $S = \{(p',q') \mid \exists n \cdot C_n[p'] \approx_{barb} C_n[q']\}$ where p' and q' range over agents defined over the value set Val. Following Theorem 3.3.2 in [94] we can show that S is a weak bisimulation and the result follows. ■

As one might expect the relation $\approx$ is not a congruence for CBS owing to the fact that it is not preserved by the summation operator. This fact is ascribed to the so called *pre-emptive* power of τ to resolve choices [74]. In CCS we define observation congruence as the largest congruence relation strictly contained in weak bisimulation and a characterisation of this observation congruence tells us that for two agents p and q to be related, any τ move from p must be matched by *at least one* τ move from Q. That is, for every possible choice made by one agent, then at least one choice must be made by the other agent and vice-versa. This helps in understanding the following definition.

Observation congruence, $\cong$, is the relation defined by $p \cong q$ if

- whenever $p \xrightarrow{w!} p'$ then $q \overset{w!}{\Longrightarrow} q'$ for some q' such that $p' \approx q'$
- whenever $p \xrightarrow{v?} p'$ then $q \overset{v?}{\Longrightarrow} q'$ for some q' such that $p' \approx q'$
  or $q \overset{\tau!v:}{\Longrightarrow} q'$ such that $p' \approx q'$
- whenever $p \xrightarrow{v:} p$ then $q \xrightarrow{v:} q$

(with symmetric conditions on q omitted).

An important consequence of this definition is that if $p \cong q$ then we know that $I(p)$ must be equal to $I(q)$. This follows from the third clause that discards must be matched with discards, and Proposition 3.1.1

Theorem 4.1.4 $(p \cong_{barb} q) : Val^{++}$ *if and only if* $(p \cong q) : Val.$

Proof We leave the reader to check that $\cong$ is preserved by all the operators in *CBS* and since $p \cong q$ trivially implies that $p \approx_{barb} q$ we conclude that $p \cong q$ implies $p \cong_{barb} q$.

Conversely, suppose $(p \cong_{barb} q) : Val^{++}$. Then $C[p + v_0!\mathbf{O}] \approx_{barb} C[q + v_0!\mathbf{O}]$ for all static contexts $C[\,\_\,]$ over Val^{++}, where v_0 is some distinguished value not in Val. It follows from Proposition 4.1.3 that $p + v_0!\mathbf{O} \approx q + v_0!\mathbf{O}$.

We now prove that $p + v_0!\mathbf{O} \approx q + v_0!\mathbf{O}$ implies $p \cong q$. As an example we show $p \xrightarrow{v:} p$ implies $q \xrightarrow{v:} q$; the remaining requirements are similar. From $p \xrightarrow{v:} p$ it follows that $p + v_0!\mathbf{O} \xrightarrow{v:} p + v_0!\mathbf{O}$ also. By the hypothesis we know that $q + v_0!\mathbf{O} \overset{v??}{\Longrightarrow} q'$ for some $q' \approx p + v_0!\mathbf{O}$. This means that q' is $q + v_0!\mathbf{O}$, otherwise $q' \overset{v_0!}{\not\Longrightarrow}$. So we have that $q + v_0!\mathbf{O} \xrightarrow{v:} q + v_0!\mathbf{O}$ which in turn implies that $q \xrightarrow{v:} q$. ■

4.2 Characterising observation congruence over simple agents

We are going to provide an axiomatic characterisation of observation congruence over the class $\mathcal{SPA}$ of agents. We then show how to use the symbolic technique to lift this characterisation to open terms. Both characterisations extend to finite CBS by using the translation of finite CBS into $\mathcal{SPA}$ of Section 3.4. The proof systems that we use to describe these characterisations are those of Section 3.3. Because noisy congruence is strictly contained in observation congruence all we need to do is to augment the axiom set $\mathcal{A}_{\mathcal{P}}$ with the axioms required to abstract away from noisy τ actions. To guide us in our search for these extra axioms we consider the three familiar τ laws of CCS, [74]:

T1 $\alpha.\tau.X =_{ccs} \alpha.X.$

T2 $\alpha.(X+\tau.Y)+\alpha.Y =_{ccs} \alpha.(X+\tau.Y)$.

T3 $X+\tau.X =_{ccs} \tau.X$.

Unfortunately the obvious versions of $T1$ and $T3$ for CBS are not sound. We have already seen, for example, that p is not, in general, weakly bisimilar to $\tau!p$ which implies that $v!\tau!p \not\cong v!p$. For $T3$, $p+\tau!p = \tau!p$, we run into difficulties when p is allowed to recieve a value v, say. For then $\tau!p$ may discard v but $p+\tau!p$ is obliged to receive it. We must look for analagous rules which respect $\cong$.

Notice that there is, in the presence of $T3$, an equipotent version of $T1$ for CCS:

$$T1' \;\; \alpha.(\tau.X+X) = \alpha.X.$$

This will stand us in good stead because we do have the property $p+\tau!p \approx p$ in CBS. Therefore we can include the axiom

$$\boxed{\textit{Tau1}: \quad e!(\tau!X+X) = e!X.}$$

The second axiom, $T2$, can be directly incorporated as

$$\boxed{\textit{Tau2}: \quad \alpha.(X+\tau!Y)+\alpha.Y = \alpha.(X+\tau!Y).}$$

The third axiom $T3$ causes us a little more trouble. We must consider under what conditions the equivalence $p+\tau!p \cong \tau!p$ will hold. We have already stated that the problem lies with p's potential to input. Bringing p out from under the τ action may unveil potential input moves. We could consider the situation where p has no input moves, wherein the rule mooted would be sound. Alternatively we could consider the case where the terms $p+\tau!p$ and $\tau!p$ are sitting in a context where discard moves are impossible, for instance

$$x \in Val?\mathbf{O} + (p+\tau!p) \cong x \in Val?\mathbf{O} + \tau!p.$$

This is indeed sound but we can give a more general description of such a situation as

$$q+p+\tau!p \cong q+\tau!p$$

where $I(p) \subseteq I(q)$. We only need to ask that no *new* input moves become possible as p is uncovered. This rule would serve perfectly well as our third axiom but for technical reasons, to be consistent with the axioms to be presented for open terms, we analyse the form of p a little further and split the third axiom into two:

$$\boxed{\textit{Tau3}: \quad X+x \in S?Z+\tau!(Y+x \in S?Z) = X+\tau!(Y+x \in S?Z) \;\; \text{if } S \subseteq I(X)}$$

and

$$\boxed{\textit{Tau4}: \quad e!X+\tau!(Y+e!X) = \tau!(Y+e!X).}$$

Let $\mathcal{A}_{\mathcal{P}\tau}$ denote the set $\mathcal{A}_{\mathcal{P}}$ along with the four τ axioms *Tau1* through *Tau4* . We will write $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = q$ if we can derive $p = q$ from the axioms in $\mathcal{A}_{\mathcal{P}\tau}$ using the proof rules of Section 3.3.

Theorem 4.2.1 *(Soundness) $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = q$ implies $p \cong q$.*

Proof Simple induction on proof of $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = q$. ■

The remainder of this section deals with the proof of the converse of this, completeness. The exposition of the completeness proof will require the use of standard form and depth of a term. In fact we use a more refined notion of standard form in which the actions are *saturated* in the sense that if an action may occur weakly, then it can occur directly, without prior τ actions.

We call a standard form, p, ***saturated*** if

(i) $p \overset{w!}{\Longrightarrow} p'$ implies $p \xrightarrow{w!} p'$.

(ii) $v \in I(p)$ and $p \overset{v?}{\Longrightarrow} p'$ implies $p \xrightarrow{v?} \overset{\varepsilon}{\Longrightarrow} p'$.

(iii) $v \in I(p)$ and $p \overset{\tau!v;}{\Longrightarrow} p'$ implies $p \xrightarrow{v?} p'$.

The side condition that $v \in I(p)$ in cases (ii) and (iii) should be clear; we anticipate that we can prove p congruent to a saturated version of p. In order to do this we must not introduce any new input actions into the saturated version of p by pulling a $v?$ action, say, out from behind a τ action. The condition $v \in I(p)$ guarantees that any input action introduced was already possible.

Lemma 4.2.2 (Derivation Lemma) *For any standard form $p \in \mathcal{SPA}$, $p \overset{w!}{\Longrightarrow} q$ implies $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + w!q$.*

Proof By induction on the length of the derivation $p \overset{w!}{\Longrightarrow} q$.

The base case is straightforward: If $p \xrightarrow{w!} q$ then $w!q$ is a summand of p because p is standard. Idempotence of $+$ gives the proof $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + w!q$.

Suppose that $p \xrightarrow{w!} p' \overset{\tau}{\Longrightarrow} q$. Then by induction we know that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p' = p' + \tau!q.$$

We also know that $w!p'$ is a summand of p so $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + w!p'$. Putting these together gives

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + w!(p' + \tau!q).$$

An application of axiom *Tau2* yields

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + w!(p' + \tau!q) + w!q$$

from which we can obtain $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + w!q$.

Now suppose that $p \xrightarrow{\tau} p' \overset{w!}{\Longrightarrow} q$. We can easily see that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + \tau!(p' + w!q)$$

by induction and using the fact that $\tau!p'$ is a summand of p. Axiom *Tau4* can be applied directly to this to give

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + w!q + \tau!(p' + w!q)$$

whence $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + w!q$. ■

Proposition 4.2.3 *Given any agent $p \in \mathcal{SPA}$, there exists a saturated standard form $\hat{p}$ such that $d(p) = d(\hat{p})$ and $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = \hat{p}$.*

Proof We know that p can be transformed into a standard form and we show that this standard form can be saturated. Essentially the proof proceeds by using the Derivation Lemma to saturate p with respect to all $w!$ derivatives.

We can use the Derivation Lemma repeatedly to ensure that for any $p \overset{w!}{\Longrightarrow} q$ we let $\hat{p}$ have $w!q$ as a summand and provability is given. Thus $\hat{p} \xrightarrow{w!} q$.

Suppose $p \overset{v?}{\Longrightarrow} q$ with $v \in I(p)$. Consider the move $p \overset{v?}{\Longrightarrow} q$ decomposed as follows (there are simpler cases, we omit them here),

$$p \overset{\tau!}{\Longrightarrow} p' \xrightarrow{v?} q'_v \overset{\varepsilon}{\Longrightarrow} q.$$

Now we know that $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + \tau!p'$ by the Derivation Lemma. We also know that we can prove $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p' = p' + x \in S?q'$ for some set S and some term q' such that $v \in S$ and q'_v is $q'[v/x]$. Combining these gives

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + \tau!(p' + x \in S?q').$$

Now axiom *Tau3* would be applicable if $S \subseteq I(p)$ but we cannot ensure this. However, by using the *Pattern* axiom we can deduce

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + \tau!(p' + x \in S \cap I(p)?q')$$

and then apply axiom *Tau3* . This gives

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + x \in S \cap I(p)?q'$$

so we include the summand $x \in S \cap I(p)?q'$ in $\hat{p}$ and note that $v \in S \cap I(p)$ so $\hat{p} \xrightarrow{v?} q_v \overset{\varepsilon}{\Longrightarrow} q$.

Finally, suppose that $p \overset{\tau!v:}{\Longrightarrow} p'$ for some p'. We show that $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + x \in S?p'$ for some set S containing v. We can break the move $p \overset{\tau!v:}{\Longrightarrow} p'$ up into $p \overset{\tau!}{\Longrightarrow} q_v \xrightarrow{v:} q_v \overset{\varepsilon}{\Longrightarrow} p'$ for some agent q_v such that $v \notin I(q_v)$. Axiom *Noisy* tells us that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \tau!q_v = \tau!(q_v + x \in S?q_v)$$

where S is the set $(Val \setminus I(q_v)) \cap I(p)$; note that $v \in S$ by hypothesis. Saturation with respect to $\tau!$ moves allows us to assume that $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + \tau!q_v$. Axiom *Tau3* will give

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + x \in S?q_v.$$

If q_v is p' we are finished. Otherwise again we can use saturation with respect to τ moves to obtain $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} q_v = q_v + \tau!p'$ and axiom *Tau2* to give $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = p + x \in S?p'$. Again, we include this summand in $\hat{p}$ and note that $\hat{p} \xrightarrow{v?} p'$. ■

We observe that, in CBS, both reception and τ-actions cause weak bisimulation to fail to be preserved by choice. This property was called *pre-emptive* power over choice in [74]. Proposition 11, Chapter 7 of [74], accounts for the pre-emptive power of τ by relating observation congruence for CCS to weak bisimulation. In Theorem 3.3.2 we proved a decomposition theorem for $\mathcal{SPA}$ which relates noisy bisimulation to noisy bisimulation, thus providing an analogous account of the pre-emptive power of reception. What we require here is a happy combination of these two decomposition theorems.

Theorem 4.2.4 *(Decomposition) Let $S = I(q) \setminus I(p)$ and $S' = I(p) \setminus I(q)$. Then $p \approx q$ iff one of the following holds:*

(i) $p + x \in S?p \cong q + x \in S'?q$ and when S and S' are both non-empty there exist p', q' such that $d(p') < d(p), d(q') < d(q)$ and $p' \approx p, q' \approx q$.

(ii) $p + x \in S?p + \tau!p \cong q + x \in S'?q$ and when S' is non-empty there exist p', q' such that $d(p') < d(p), d(q') < d(q)$ and $p' \approx p, q' \approx q$.

(iii) $p + x \in S?p \cong q + x \in S'?q + \tau!q$ and when S is non-empty there exist p', q' such that $d(p') < d(p), d(q') < d(q)$ and $p' \approx p, q' \approx q$.

Proof The *if* direction is standard. So suppose $p \approx q$. There are three cases.

- Suppose there exists a p^τ such that $p \xrightarrow{\tau!} p^\tau$ and for each q' such that $q \overset{\tau!}{\Longrightarrow} q'$ we have $p^\tau \not\approx q'$. In this case we show that (iii) holds.

  We first notice that

$$\begin{array}{rcl} I(p+x\in S?p) & = & I(p)\cup I(x\in S?p) \\ & = & I(p)\cup(I(q)\setminus I(p)) \\ & = & I(p)\cup I(q) \\ & = & I(q)\cup(I(p)\setminus I(q)) \\ & = & I(q+x\in S'?q+\tau!q). \end{array}$$

  This means that $p+x\in S?p \xrightarrow{v:} p+x\in S?p$ if and only if $q+x\in S'?q+\tau!q \xrightarrow{v:} q+x\in S'?q+\tau!q$.

  Suppose $p+x\in S?p \xrightarrow{v!} p'$. Then $p \xrightarrow{v!} p'$. Because $p\approx q$ we know that $q \overset{v!}{\Longrightarrow} q'$ for some q' such that $p'\approx q'$. Therefore $q+x\in S'?q+\tau!q \overset{v!}{\Longrightarrow} q'$ also. Similarly if $q+x\in S'?q+\tau!q \xrightarrow{v!} q'$.

  Suppose $p+x\in S?p \xrightarrow{\tau!} p'$. Then $p \xrightarrow{\tau!} p'$. So we know that $q \overset{\varepsilon}{\Longrightarrow} q'$ for some q' such that $p'\approx q'$. Therefore $q+x\in S'?q+\tau!q \overset{\tau!}{\Longrightarrow} q'$. Conversely suppose $q+x\in S'?q+\tau!q \xrightarrow{\tau!} q'$. Here there are two possibilities: If $\tau!q \xrightarrow{\tau!} q' \equiv q$ then we have $p \overset{\tau!}{\Longrightarrow} p^\tau$ with $p^\tau \approx q$. Otherwise we must have $q \xrightarrow{\tau!} q'$. In which case we know that $p^\tau\approx q$ so we have a p' such that $p^\tau \overset{\varepsilon}{\Longrightarrow} p'$ with $p'\approx q'$. But we also have that $p \xrightarrow{\tau!} p^\tau$. Therefore $p \overset{\tau!}{\Longrightarrow} p'$.

  Suppose $p+x\in S?p \xrightarrow{v?} p'$.

  If $p \xrightarrow{v?} p'$ we know that $q \overset{v??}{\Longrightarrow} q'$ for some q' such that $p'\approx q'$. Therefore $\tau!q \overset{\tau!v??}{\Longrightarrow} q'$ and so $q+x\in S'?q+\tau!q \overset{v?}{\Longrightarrow} q'$ or $q+x\in S'?q+\tau!q \overset{\tau!v:}{\Longrightarrow} q'$.

  On the other hand, if it is the case that $x\in S?p \xrightarrow{v?} p'\equiv p$, so that $v\in S$, we then know that $v\notin I(p)$ and $p \xrightarrow{v:} p$. Because $p\approx q$ we know that there exists a q' such that $q \overset{v??}{\Longrightarrow} q'$ and $p\approx q'$. Thus $\tau!q \overset{\tau!v??}{\Longrightarrow} q'$ which gives $q+x\in S'?q+\tau!q \overset{v?}{\Longrightarrow} q'$ or $q+x\in S'?q+\tau!q \overset{\tau!v:}{\Longrightarrow} q'$.

  If $q+x\in S'?q+\tau!q \xrightarrow{v?} q'$ where $q \xrightarrow{v?} q'$ then there exists a p' such that $p \overset{v??}{\Longrightarrow} p'$ and $p'\approx q'$. If $p \overset{v?}{\Longrightarrow} p'$ or $p \overset{\tau!v:}{\Longrightarrow} p'$ then we have a match, otherwise we know that $p \xrightarrow{v:} p \overset{\varepsilon}{\Longrightarrow} p'$ which entails that $v\in I(q)\setminus I(p)$. Thus we have $x\in S?p \xrightarrow{v?}\overset{\varepsilon}{\Longrightarrow} p'$.

  It only remains to check the case $x\in S'?q \xrightarrow{v?} q'$. We see that $v\notin I(q)$, $v\in I(p)$ and, in this case, q' must be q. Since $p\approx q$ and $q \overset{v:}{\Longrightarrow} q$ we know that there exists a p' such that $p \overset{v??}{\Longrightarrow} p'$ with $p'\approx q$ but since $v\in I(p)$ we must have $p \overset{v?}{\Longrightarrow} p'$ or $p \overset{\tau!v:}{\Longrightarrow} p'$.

  This completes the proof that $p+x\in S?p \cong q+x\in S'?q+\tau!q$. Now suppose S is non-empty. We have to find p', q' such that $d(p')<d(p)$, $d(q')<d(q)$ and $p'\approx p$, $q'\approx q$. We already know that $p^\tau\approx q$ and transitivity gives $p^\tau\approx p$; so the required p' is p^τ. To find the required q' let v_0 be any value from the non-empty set S. Since $v_0\notin I(p)$ this means $p \xrightarrow{v_0:} p$. So there exists a q' such that $q \overset{v_0??}{\Longrightarrow} q'$ and $p\approx q'$; by transitivity this means $q'\approx q$. But $v_0\in I(q)$ which forces $q \overset{v_0?}{\Longrightarrow} q'$ or $q \overset{\tau!v_0:}{\Longrightarrow} q'$; either way $d(q')<d(q)$ holds.

- Suppose there exists a q^τ such that $q \xrightarrow{\tau!} q^\tau$ and for each p' such that $p \overset{\tau!}{\Longrightarrow} p'$ we have $p'\not\approx q^\tau$.

  This is a symmetric version of the first case and in a similar manner one can show that (ii) holds.

- If neither of these two conditions apply then one can show by simple checking that case (i) holds.

■

Theorem 4.2.5 *(Completeness) For all agents $p, q \in \mathcal{SPA}$,*

$$p \cong q \text{ implies } \mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} p = q$$

Proof The proof is by induction on the combined depth of p and q.

Because of Lemma 4.2.3 we can assume that p and q can be transformed to saturated standard forms

$$\sum_I e_i!p_i + \sum_J x \in S_j?t_j, \quad \sum_K e_k!q_k + \sum_L x \in S_l?u_l$$

respectively. It is sufficient, because of saturation, to prove that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \sum_I e_i!p_i = \sum_K e_k!q_k$$

and

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \sum_J x \in S_j?t_j = \sum_L x \in S_l?u_l$$

and as an example we consider the latter. To establish this it is sufficient, by symmetry, to prove for an arbitrary $j \in J$ that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} x \in S_j?t_j + \sum_L x \in S_l?u_l = \sum_L x \in S_l?u_l.$$

For each $v \in S_j$ we know that $p \xrightarrow{v?} t_j[v/x]$. This means $v \in I(p)$ and, since $p \cong q$, then $v \in I(q)$. We also know, because $p \cong q$ and q is saturated, that $q \xrightarrow{v?} u_l[v/x] \overset{\varepsilon}{\Longrightarrow} q_l^v$ for some $l \in L$ such that $v \in S_l$ and $t_j[v/x] \approx q_l^v$. Let $S_l^j = \{v \in S_j \cap S_l \mid q_l^v \approx t_j[v/x]\}$. This gives a *finite* partition $\{S_l^j\}_{l \in L}$ of S_j such that $S_l^j \subseteq S_l$ for each $l \in L$. Then, by the idempotency of $+$ and the axiom *Pattern* it is sufficient to show, for each $l \in L$, that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} x \in S_l^j?t_j + x \in S_l^j?u_l = x \in S_l^j?u_l.$$

This can be inferred from the rule cl-P-INPUT and *Tau2* if we can prove for each $v \in S_l^j$

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \tau!(\tau!t_j[v/x] + u_l[v/x]) = \tau!u_l[v/x].$$

So let us fix a particular $v \in S_l^j$ and see how this can be inferred. We know that $t_j[v/x] \approx q_l^v$ so from this we will show that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \tau!t_j[v/x] = \tau!q_l^v$$

and the result will follow by the Derivation Lemma and rule TAU.

For convenience let p, q denote $t_j[v/x], q_l^v$ respectively. We now apply Theorem 4.2.4 to get one of three possibilities

(i) $p + x \in U?p \cong q + x \in V?q$

(ii) $p + x \in U?p + \tau!p \cong q + x \in V?q$

(iii) $p + x \in U?p \cong q + x \in V?q + \tau!q$

where $U = I(q) \setminus I(p)$ and $V = I(p) \setminus I(q)$. We show how to deal with case (iii) and leave cases (i) and (ii) to the reader. We have two eventualities to consider.

1. $U = \emptyset$
   Here we have $p \cong q + x \in V?q + \tau!q$ and we can use induction to obtain $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \tau!p = \tau!(q + x \in V?q + \tau!q)$. Now we can apply the *Noisy* scheme to obtain

   $$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \tau!p = \tau!(q + x \in V?q + \tau!(q + x \in V?q))$$

   from which $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \tau!p = \tau!(q + x \in V?q)$ follows by *Tau1* . Another appplication of *Noisy* gives the required result.

2. $U \neq \emptyset$
   Here we have $p + x \in U?p \cong q + x \in V?q + \tau!q$ and in this case we cannot apply induction immediately as the combined depth of the terms has not decreased. But Thereom 4.2.4 tells us that there exists p', q' such that $d(p') < d(p)$ and $d(q') < d(q)$ such that $p' \approx p$ and $q' \approx q$. Suppose without loss of generality that $d(p) \leq d(q)$. Then, since $\tau!p \cong \tau!p'$, we can use induction to obtain $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \tau!p = \tau!p'$. A simple application of the cl-P-INPUT rule gives $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} x \in U?p = x \in U?p'$. This in turn implies that $p + x \in U?p' \cong q + x \in V?q + \tau!q$ and here we can apply induction since the combined size has decreased. So we obtain

   $$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \tau!(p + x \in U?p') = \tau!(q + x \in V?q + \tau!q).$$

   Using the fact that $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} x \in U?p = x \in U?p'$ we get

   $$\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \tau!(p + x \in U?p) = \tau!(q + x \in V?q + \tau!q)$$

   from which the required $\mathcal{A}_{\mathcal{P}\tau} \vdash_{cl} \tau!p = \tau!q$ follows by applications of the *Noisy* and *Tau1* axioms.

■

4.3 Characterising weak bisimulation symbolically

Once again we turn to the symbolic approach to show how the proof system of the previous system may be made finitary. The tools we use should now be familiar: The abstract operational semantics of Figure 3.5 form a symbolic graph for $\mathcal{SPA}$ upon which weakened symbolic transitions $\overset{b,\alpha}{\Longrightarrow}$ are used to define weak symbolic bisimulations. The notions from Section 3.3.1 of t-uniformity and the relativised input set $I(b,t)$ for open terms will also be prevalent.

The characterisation we present in this section extends the proof system of Section 3.3.1 by adding appropriate laws for manipulating τ-prefixes. We can see from the previous section that the extra τ-laws that we require will be the four axioms *Tau1* to *Tau4* , adapted for open terms. A quick inspection tells us that, of these four, only *Tau3* ,

$$X + x \in S?Z + \tau!(Y + x \in S?Z) = X + \tau!(Y + x \in S?Z) \;\text{ if } S \subseteq I(X)$$

cannot be used for open terms because of the side-condition $S \subseteq I(X)$. We use the approach of Section 3.3.1 and invoke the uniformity condition. Axiom *Tau3* can be expressed for open terms as follows:

$$b \triangleright X + x \in S?Z + \tau!(Y + x \in S?Z) = X + \tau!(Y + x \in S?Z)$$

if b is X-uniform and $S \subseteq I(b,X)$.

To recap, we will write $\mathcal{A}_{\mathcal{P}\tau}$ for the set of axioms $\mathcal{A}_{\mathcal{P}}$ along with the four τ-laws, *Tau1* to *Tau4* where *Tau3* is defined over open terms. We will also write $\mathcal{A}_{\mathcal{P}\tau} \vdash b \triangleright t = u$ to mean that $b \triangleright t = u$ may be deduced from the axioms in $\mathcal{A}_{\mathcal{P}\tau}$ using the rules of Section 3.3.1.

Theorem 4.3.1 *(Soundness) If* $\mathcal{A}_{\mathcal{P}\tau} \vdash b \triangleright t = u$ *then* $(\forall \delta \cdot \delta \models b$ *implies* $t\delta \cong u\delta)$

Proof This is simply a matter of checking that each axiom and rule is sound with respect to the appropriate semantic equivalence. As an example we check *Tau3* with respect to $\cong$.

Given δ such that $\delta \models b$. b is t-uniform so $I(b,t) = I(t\delta)$. We need to show that $t\delta + x \in S?u_1\delta + \tau!(u_2\delta + x \in S?u_1\delta) \cong t\delta + \tau!(u_2\delta + x \in S?u_1\delta)$ for any terms u_1 and u_2 given that $S \subseteq I(t\delta)$. The only interesting moves to match are the discards:

$$I(t\delta + x \in S?u_1\delta + \tau!(u_2\delta + x \in S?u_1\delta)) = S \cup I(t\delta) = I(t\delta)$$

and

$$I(t\delta + \tau!(u_2\delta + x \in S?u_1\delta)) = I(t\delta).$$

■

Completeness is obtained using symbolic semantics. We recall the symbolic graph defined by the symbolic operation semantics in Figure 3.5. Using the relation $\longmapsto$ we define the weakened symbolic transition relation as the least relation satisfying the following rules:

$t \overset{\mathbf{tt},\varepsilon}{\Longrightarrow} t$

$t \overset{b,\alpha}{\longmapsto} t'$ implies $t \overset{b,\alpha}{\Longrightarrow} t'$

$t \overset{b,\tau!}{\longmapsto} t' \overset{b',\alpha}{\Longrightarrow} t''$ implies $t \overset{b\wedge b',\alpha}{\Longrightarrow} t''$

$t \overset{b,\alpha}{\Longrightarrow} t' \overset{b',\tau!}{\longmapsto}$ implies $t \overset{b\wedge b',\alpha}{\Longrightarrow} t''$.

We must remark that weak symbolic transitions do not enjoy the property that $fv(b) \subseteq fv(t)$ whenever $t \overset{b,\alpha}{\Longrightarrow} t'$, which is demanded of strong symbolic transitions. However, this causes no real problems for the ensuing theory of because CBS induces saturated graphs which support α-conversion. Thus to avoid clashes of free and bound variables we can assume that if $x \in fv(b)$ is bound by α, then $x \notin fv(t)$.

The definition of ***weak symbolic bisimulation*** is the obvious one, *viz.* the weakened version of *patterned noisy symbolic bisimulations* of Section 3.3.1. Suppose $\mathbf{S} = \{S^b\}$ is a boolean indexed family of relations. Define $\mathcal{WSB}(\mathbf{S})$ to be the family of relations such that

$(t,u) \in \mathcal{WSB}(\mathbf{S})^b$ if whenever (omitting symmetric conditions on u)

- $t \overset{b_1,e!}{\longmapsto} t'$ there exists a $b \wedge b_1$-partition, B, and for each $b' \in B$ there exists a $u \overset{b_2,\hat{e}'!}{\Longrightarrow} u'$ such that $b' \models b_2$, $b' \models \hat{e} = \hat{e}'$ and $(t',u') \in S^{b'}$
- $t \overset{b_1,x\in S?}{\longmapsto} t'$ there exists a variable z such that $z \notin fv(b,t,u)$ and a $b \wedge b_1 \wedge z \in S$-partition, B, and for each $b' \in B$ there exists $u \overset{b_2,y\in S'?}{\Longrightarrow} u'$ for some y and some S' such that $b' \models b_2$, $b' \models z \in S'$ and $(t'[z/x], u'[z/y]) \in S^{b'}$ *or* there exists $u \overset{b_2,S':}{\Longrightarrow} u$ such that $b' \models b_2$, $b' \models z \in S'$ and $(t'[z/x], u) \in S^{b'}$.
- $t \overset{b_1,S:}{\longmapsto} t$ there exists a variable z such that $z \notin fv(b,t,u)$ and a $b \wedge b_1 \wedge z \in S$-partition, B, and for each $b' \in B$ there exists $u \overset{b_2,y\in S'?}{\Longrightarrow} u'$ for some y and some S' such that $b' \models b_2$, $b' \models z \in S'$ and $(t, u'[z/y]) \in S^{b'}$ *or* there exists $u \overset{b_2,S':}{\Longrightarrow} u$ such that $b' \models b_2$, $b' \models z \in S'$ and $(t,u) \in S^{b'}$.

Here we generalise the notation used previously by letting $\hat{e}$ denote ε if e is τ and e otherwise. We call $\{S^b\}$ a weak symbolic bisimulation if $S^b \subseteq \mathcal{WSB}(\mathbf{S})^b$ for each b and denote the largest such $\mathbf{S}$ by $\{\approx^b\}$. Once again we now use the definition of $\approx^b$ to define $\cong^b$ the largest congruence contained in $\approx^b$:

$t \cong^b u$ if whenever

- $t \overset{b_1,e!}{\longmapsto} t'$ there exists a $b \wedge b_1$-partition, B, and for each $b' \in B$ there exists a $u \overset{b_2,e'!}{\Longrightarrow} u'$ such that $b' \models b_2$, $b' \models e = e'$ and $t' \approx^{b'} u'$
- $t \overset{b_1,x \in S?}{\longmapsto} t'$ there exists a variable z such that $z \notin fv(b,T,U)$ and a $b \wedge b_1 \wedge z \in S$-partition, B, such that for each $b' \in B$ there exists a $u \overset{b_2,y \in S'?}{\Longrightarrow} u'$ for some y, such that $b' \models b_2$, $b' \models z \in S'$ and $t'[z/x] \approx^{b'} u'[z/y]$ *or* there exists a $u \overset{b_2,\tau!S':}{\Longrightarrow} u'$ such that $b' \models b_2$, $b' \models z \in S'$ and $t'[z/x] \approx^{b'} u'$.
- $t \overset{b_1,S:}{\longmapsto} t$ there exists a $u \overset{b_2,S':}{\longmapsto} u$ such that $b \wedge b_1 \models b_2$ and $S \subseteq S'$

(Symmetric conditions on u are omitted).

It is a simple to matter to adapt the proof of Theorem 3.3.6 to weak terms to yield:

Theorem 4.3.2 *For any terms t and u,*

$$t \cong^b u \textit{ iff } (\forall \delta \cdot \delta \models b \textit{ implies } t\delta \cong u\delta).$$

This and Theorem 4.2.4 give a decomposition theorem for *open* terms directly.

Theorem 4.3.3 *(Decomposition) If t and u are standard forms and $t \approx^b u$ then there exists a b-partition, B, such that for each $b' \in B$, b' is both t and u-uniform and one of the following holds, where $S = I(b',u) \setminus I(b',t)$ and $S' = I(b',t) \setminus I(b',u)$.*

(i) $t + x \in S?t \cong^{b'} u + x \in S'?u$ *and when S and S' are both non-empty there exist t', u' such that $d(t') < d(t), d(u') < d(u)$ and $t' \approx^{b'} t, u' \approx^{b'} u$.*

(ii) $t + x \in S?t + \tau!t \cong^{b'} u + x \in S'?u$ *and when S' is non-empty there exist t', u' such that $d(t') < d(t), d(u') < d(u)$ and $t' \approx^{b'} t$, $u' \approx^{b'} u$.*

(iii) $t + x \in S?t \cong^{b'} u + x \in S'?u + \tau!u$ *and when S is non-empty there exist t', u' such that $d(t') < d(t), d(u') < d(u)$ and $t' \approx^{b'} t$, $u' \approx^{b'} u$.*

Proof Simple adaption of the proof of Theorem 3.3.7. ■

The next step towards the completeness proof is to develop the notion of saturation for open terms. Unfortunately there are inconvenient side conditions for saturation in the open term proof system which make it impractical to work with a notion of a saturated form. Instead we present a weaker form of saturation. First however, we make a few comments about abstract discard moves.

Given a term $t_? \equiv \sum_{i \in I_?} b_i \gg x_i \in S_i?t_i$. Consider how a possible discard transition $t_? \overset{b,S:}{\longmapsto} t_?$ may occur. It is not difficult to see that b must be of the form $\bigwedge_{i \in K} \neg b_i$ and S of the form $\bigcap_{i \in I_? \setminus K}(Val \setminus S_i)$ for some $K \subseteq I_?$. Therefore any discard $t_? \overset{b,S:}{\longmapsto} t_?$ can be described by a set $K \subseteq I_?$; we call such a set the ***discard index*** of $t_? \overset{b,S:}{\longmapsto} t_?$.

Discard moves from general standard forms still have a discard index. If t is a standard form with indexing sets $I_!$ and $I_?$, and if $t \overset{b,S:}{\longmapsto} t$ then there exists a set $K \subseteq I_?$ such that $b \models \bigwedge_{i \in K} \neg b_i$ and $S = \bigcap_{i \in I_? \setminus K}(Val \setminus S_i)$.

The following proofs make use of these derivable variants of the axiom *Tau3*

$$t + b' \gg \tau!(b \gg x \in S?u) = t + b' \gg \tau!(b \gg x \in S?u) + b \gg x \in S?u$$

(if $b \models b'$, b is t-uniform and $S \subseteq I(b,t)$)

and the P-INPUT rule

$$\frac{b \wedge x \in S \rhd \sum_I b_i \gg \tau!t_i = \sum_J b_j \gg \tau!u_j}{b \rhd \sum_I b_i \gg x \in S?t_i = \sum_J b_j \gg x \in S?u_j} \quad \text{if } x \notin fv(b,b_i,b_j).$$

We call these *Tau3*$^{\gg}$ and P-INPUT$^{\gg}$ respectively.

Lemma 4.3.4 (Derivation Lemma) *For any term t*

(i) If $t \overset{b,e!}{\Longrightarrow} t'$ then $\mathcal{A}_{\mathcal{P}\tau} \vdash t = t + b \gg e!t$.

(ii) If b is t-uniform, $S \subseteq S' \cap I(b,t)$, and $b \models b_1 \wedge b_2$ where $t \overset{b_1,\varepsilon}{\Longrightarrow} \overset{b_2,x\in S'}{\longmapsto} t'$ then we have that $\mathcal{A}_{\mathcal{P}\tau} \vdash t = t + b \gg x \in S?t'$.

(iii) If b is t-uniform, $S \subseteq S' \cap I(b,t)$, and $b \models b'$ where $t \overset{b',\tau!S':}{\Longrightarrow} t'$ then $\mathcal{A}_{\mathcal{P}\tau} \vdash t = t + b \gg x \in S?t'$ for some $x \notin fv(b,t)$.

Proof

(i) This is straightforward. We use induction on the derivation of $t \overset{b,e!}{\Longrightarrow} t'$.

(ii) We assume, without loss of generality, that t is a standard form. **Case** $t \overset{b_2,x\in S'?}{\longmapsto} t'$. So we know $b_2 \gg x \in S'?t'$ is a summand of t.

Idempotence	$\mathcal{A}_{\mathcal{P}\tau} \vdash t = t + b_2 \gg x \in S'?t'$
$b \models b_2,\ S \subseteq S'$	$\mathcal{A}_{\mathcal{P}\tau} \vdash t = t + b \gg x \in S?t'$.

Case $t \overset{b_1,\tau!}{\Longrightarrow} u \overset{b_2,x\in S'?}{\longmapsto} t'$.

Suppose that $u_? \equiv \sum_{I_?} b_i \gg x_i \in S_i?u_i$. For each $K \subseteq I_?$ let

$$b_K = \bigwedge_{i\in K} b_K \wedge \bigwedge_{i\in I_?\setminus K} \neg b_i.$$

Define $B_u = \{b \wedge b_K \mid K \subseteq I_?\}$. Clearly then B_u is a u-uniform partition of b. Choose any $b_u (= b \wedge b_K \neq \mathbf{ff}) \in B_u$. We know that $b_u \models b \models b_2$ so b_2 must be b_{i_0} for some $i_0 \in K$. This means that $S' = S_{i_0} \subseteq I(b_u,u)$. Therefore

$$\mathcal{A}_{\mathcal{P}\tau} \vdash u = u + b_u \gg x \in S'?t'.$$

This is true for each $b_u \in B_u$ so we can add to get

$$\mathcal{A}_{\mathcal{P}\tau} \vdash u = u + \sum_{B_u} b_u \gg x \in S'?t'.$$

Thus, by manipulating the boolean guards using Proposition 3.2.3 and remembering that B_u is a b partition, we get

$$\mathcal{A}_{\mathcal{P}\tau} \vdash u = u + b \gg x \in S'?t',$$

whence

$$\mathcal{A}_{\mathcal{P}\tau} \vdash u = u + b \gg x \in S?t'.$$

Using part (i) we know that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash t = t + b_1 \gg \tau!(u + b \gg x \in S?t').$$

Recall that $b \models b_1$, b is t-uniform and $S \subseteq I(b,t)$ so we can apply *Tau3*$^{\gg}$ to get the result.

(iii) Again, we assume, without loss of generality, that t is a standard form. We know that $t \overset{b',\tau!S':}{\Longrightarrow} t'$ so suppose

$$t \overset{b_1,\tau!}{\Longrightarrow} u \overset{b_2,S':}{\longmapsto} u \overset{b_3,\varepsilon}{\Longrightarrow} t'$$

where $b' = b_1 \wedge b_2 \wedge b_3$. Suppose also that $u_? \equiv \sum_{I_?} b_i \gg x \in S_i?u_i$. Then

$$b_2 = \bigwedge_{j \in J} \neg b_j \text{ and } S' = \bigcap_{j \in I_? \setminus J} (Val \setminus S_j)$$

for some discard index $J \subseteq I_?$. We let $B_u = \{b \wedge b_K \mid K \subseteq I_?\}$ be a u-uniform b partition and observe that whenever $j \in K \cap J$ we have that $b \wedge b_K \models b_j$ and $b \wedge b_K \models b_2 \models \neg b_j$. Reading this contrapositively we have that $b \wedge b_K \neq \mathbf{ff}$ implies $K \cap J = \emptyset$.

Our intention is to prove

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b \wedge b_K \rhd \tau!u = \tau!(u + x \in S?u)$$

by applying axiom *P-Noisy* (or ABSURD when $b \wedge b_K = \mathbf{ff}$) to u for each $b \wedge b_K$. In order to do this we need to show that $S \cap I(b \wedge b_K, u) = \emptyset$ whenever $b \wedge b_K \neq \mathbf{ff}$.

Suppose then that $b \wedge b_K \neq \mathbf{ff}$ and suppose for contradiction that $v \in S \cap I(b \wedge b_K, u)$. This means that $v \in S$ and $v \in S_{j_0}$ for some $j_0 \in K$. But $v \in S \subseteq S'$ implies that $v \in S' = \bigcap_{j \in I_? \setminus J}(Val \setminus S_j)$, that is $v \notin S_j$ for each $j \in I_? \setminus J$. Therefore $j_0 \notin I_? \setminus J$ and we conclude that $j_0 \in J$, which contradicts $K \cap J = \emptyset$.

We can now apply axiom *P-Noisy* (ABSURD) for each $b \wedge b_K$ in B_u and then use CASE to obtain

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b \rhd \tau!u = \tau!(u + x \in S?u).$$

Boolean manipulation and part (i) gives

$$\mathcal{A}_{\mathcal{P}\tau} \vdash t = t + b \gg \tau!(u + b \gg x \in S?u).$$

So an application of axiom *Tau3*$^{\gg}$ yields

$$\mathcal{A}_{\mathcal{P}\tau} \vdash t = t + b \gg x \in S?u.$$

The result follows easily now; if u is t' we are done, otherwise we use part (i) to give

$$\mathcal{A}_{\mathcal{P}\tau} \vdash t = t + b \gg x \in S?(u + \tau!t')$$

and apply axiom *Tau2* to finish.

■

Theorem 4.3.5 *(Completeness)*

$$t \cong^b u \textit{ implies } \mathcal{A}_{\mathcal{P}\tau} \vdash b \rhd t = u.$$

Proof We proceed by induction on the sum of the depths of t and u. Recall from the proof of Theorem 3.3.8 that, given a standard form

$$t \equiv \sum_{I_!} c_i \gg e_i!t_i + \sum_{I_?} c_i \gg z \in S_i?t_i$$

where $z \notin fv(b, T, U)$, it is possible to manipulate this form so that the we can assume the property that:

whenever $t \overset{c_i,z\in S_i?}{\longmapsto} t_i$ we get a matching partition such that each element is of the form $b' \wedge z \in S_i$ where the b' form a $b \wedge c_i$-partition, B.

Moreover, this partition can be made u-uniform by defining

$$B_u = \{b' \wedge d_K \mid b' \in B,\ K \subseteq J_?\}$$

where $J_?$ is the indexing set of u and d_K is defined to be

$$\bigwedge_{j\in K} d_j \wedge \bigwedge_{j\in J_?\setminus K} \neg d_j.$$

Therefore we assume that this property holds of both t and u which have standard forms

$$\sum_{i\in I_!} c_i \gg e_i!t_i + \sum_{i\in I_?} c_i \gg z \in S_i?t_i$$

and

$$\sum_{j\in J_!} d_j \gg e_i!u_j + \sum_{j\in J_?} d_j \gg z \in S_j?u_j$$

respectively.

It is sufficient, due to symmetry, to prove for every transition $t \overset{b',\alpha}{\longmapsto} t'$ that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b \rhd b' \gg \alpha.t' + u = u$$

where α is of the form $e!$ or $x \in S?$ We show how to deal with the latter, the former being slightly easier.

Fix $i \in I_?$ and consider $t \overset{c_i,z\in S_i?}{\longmapsto} t_i$. We know that there exists a u-uniform, $b \wedge c_i \wedge z \in S_i$-partition B_u such that each $b_u \in B_u$ is of the form $b' \wedge z \in S_i$ where the $\{b'\}$ form a $b \wedge c_i$ partition. Furthermore, each b' is of the form $b'_0 \wedge d_K$ for some $K \subseteq J_?$. For each such b_u there exists a $u \overset{d_1,\varepsilon}{\Longrightarrow} \overset{d_2,z\in S?}{\longmapsto} u'' \overset{d_3,\varepsilon}{\Longrightarrow} u'$ with $b' \models d_1 \wedge d_2 \wedge d_3$ (in this case we write $b'(?)$ and we define db' to be $d_1 \wedge d_2 \wedge d_K$) or a $u \overset{d,\tau!S:}{\Longrightarrow} u'$ with $b' \models d$, $S_i \subseteq S$ and $t_i \approx^{b_u} u'$ (in this case we write $b'(\tau:)$ and we define db' to be $d \wedge d_K$). Note that $b_u \models db'$, $z \notin fv(db')$ and db' is u-uniform. Suppose that we can prove

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b_u \rhd \tau!t_i + \tau!u' \quad = \quad \tau!u' \tag{4.1}$$

for each b_u. If u'' differs from u' then it follows from Lemma 4.3.4, Part (i) that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b_u \rhd u'' = u'' + \tau!u'.$$

From which we deduce

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b_u \rhd \tau!t_i + \tau!u'' = \tau!u''$$

by using rule TAU and axiom *Tau2* . Let

$$u^\tau = \sum_{b'(?)} db' \gg \tau!u'' + \sum_{b'(\tau:)} db' \gg \tau!u'$$

and let

$$u^? = \sum_{b'(?)} db' \gg z \in S_i?u'' + \sum_{b'(\tau:)} db' \gg z \in S_i?u'.$$

Then, assuming (4.1), we can prove that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b_u \rhd c_i \gg \tau!t_i + u^\tau = u^\tau$$

for each $b_u \in B_u$. An application of CASE will give us that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b \wedge c_i \wedge z \in S_i \rhd c_i \gg \tau!t_i + u^\tau = u^\tau$$

and then P-INPUT$^{\gg}$ yields

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b \wedge c_i \rhd c_i \gg z \in S_i?t_i + u^? = u^?.$$

We know that, because $I(b',u) = I(db',u)$, provided we can establish the hypothesis that $S_i \subseteq I(b',u)$, Lemma 4.3.4, Part (ii), gives us that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash u = u + db' \gg z \in S_i?u'$$

in the case that $b'(\tau:)$, and that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash u = u + db' \gg z \in S_i?u''$$

for $b'(?)$. Adding these together we establish $\mathcal{A}_{\mathcal{P}\tau} \vdash u = u + u^?$ whence

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b \wedge c_i \rhd c_i \gg z \in S_i?t_i + u = u.$$

Application of GUARD will then give the result required.

So let us establish the hypotheses of Lemma 4.3.4. Now $b' \models b$ so $t \cong^{b'} u$. We consider the discard from u with discard index $J_? \setminus K$ (remember $b' = b'_0 \wedge d_K$), *viz*,

$$u \overset{b_{dc},S_{dc}:}{\longmapsto} u$$

where $b_{dc} = \bigwedge_{j \in J_? \setminus K} \neg d_j$ and $S_{dc} = \bigcap_{j \in K}(Val \setminus S_j)$. This must be matched by $t \overset{b^*,S^*:}{\longmapsto} t$ where

$$b' \wedge b_{dc} \models b^* \text{ and } S_{dc} \subseteq S^*.$$

We know that $b' \models c_i$ and therefore $b^* \not\models \neg c_i$. This means that i is not in the discard index of $t \overset{b^*,S^*:}{\longmapsto} t$ which in turn means that $S_i \subseteq (Val \setminus S^*)$. But $Val \setminus S^* \subseteq Val \setminus S_{dc} = \bigcup_{j \in K} S_j = I(b',u)$ so we have $S_i \subseteq I(b',u)$.

We also fulfil our obligation in proving (4.1):

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b_u \rhd \tau!t_i + \tau!u' = \tau!u'.$$

For convenience let t' denote t_i. We know that $t' \approx^{b_u} u'$ so we can apply the Decomposition Theorem 4.3.3 to obtain a b_u-partition, B' which is both t' and u'-uniform such that for each $b'' \in B'$ one of three cases holds. We aim to prove that

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b'' \rhd \tau!t' = \tau!u'$$

for each $b'' \in B'$. We consider the case

$$t' + x \in S?t' \cong^{b''} u' + x \in S'?u' + \tau!u'$$

($S = I(b'',u') - I(b'',t')$, $S' = I(b'',t') - I(b'',u')$) as an example, the other cases can be dealt with similarly. If S is empty then we have that

$$t' \cong^{b''} u' + x \in S'?u' + \tau!u'$$

so induction and TAU give

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b'' \rhd \tau!t' = \tau!(u + x \in S'?u' + \tau!u').$$

We obtain the result using *P-Noisy* and *Tau1* . Assume then that S is not empty. We cannot apply induction immediately because the joint depths of the terms has not decreased. However, the Decomposition Theorem 4.3.3 gives terms t'' and u'' such that $d(t'') < d(t')$, $d(u'') < d(u')$, $t'' \approx^{b''} t'$ and $u'' \approx^{b''} u'$. Without loss of generality we assume that $d(t') \leq d(u')$. By induction it follows that $\mathcal{A}_{\mathcal{P}\tau} \vdash b'' \rhd \tau!t' = \tau!t''$, whence $\mathcal{A}_{\mathcal{P}\tau} \vdash b'' \rhd z \in S?t' = z \in S?t''$ by P-INPUT. It is clear that

$$t' + x \in S?t'' \cong^{b''} u' + x \in S'?u' + \tau!u'$$

and induction is applicable here yielding

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b'' \rhd t' + x \in S?t'' = u' + x \in S'?u' + \tau!u'.$$

Using the previous result we can substitute t' for t'' and apply TAU and axiom *P-Noisy* to get

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b'' \rhd \tau!t' = \tau!(u' + x \in S'?u' + \tau!u').$$

the result follows as in the case where S is empty. Application of CASE and Idempotence will now yield

$$\mathcal{A}_{\mathcal{P}\tau} \vdash b_u \rhd \tau!t' + \tau!u' = \tau!u'.$$

■

This finishes our completeness proof. The result can be lifted to cope with finite CBS by using the codings of Section 3.4 in exactly the same way as the proof systems for strong noisy congruence. This provides CBS with a powerful equational theory of observation congruence. Recall that the congruence we considered was derived from barbed bisimulations using an early semantics for CBS — we seem to have neglected the late semantics entirely. Therefore we end this Chapter with some comments about late bisimulations in CBS.

4.4 A late semantics for CBS?

We consider what the late semantics for CBS might be and argue that they do not make good computational sense in this paradigm. We recall from Chapter 2 that the move to a late semantics involved breaking up a reception $c?x.t \xrightarrow{c?v} t[v/x]$ into two parts: First we consider the move

$$c?x.t \xrightarrow{c?} (x)t$$

to a ?-abstraction, that is a function from Val to closed terms. Then we consider the behaviour of $(x)t$ after application, that is the behaviour of the term $t[v/x]$, for each value $v \in Val$. So a late semantics for CBS might follow a similar approach albeit complicated by the use of pattern-matching. The move $x \in S?t \xrightarrow{v?} t[v/x]$ would have to be broken up into

$$x \in S?t \xrightarrow{?} (x \in S)t$$

where the ?-abstraction is no longer a function from Val to closed terms but from the possibly smaller domain S. We have written $(x \in S)t$ to highlight this fact. For convenience we will annotate the transition label $x \in S?t \xrightarrow{S?} (x \in S)t$ with the set S also.

Now, suppose that we had developed a late semantics with abstractions of the form $(x \in S)t$. We consider the following processes:

$$p \Leftarrow x \in \{1,2\}?t \quad \text{and} \quad q \Leftarrow y \in \{2,3\}?u.$$

What would the (late) transitions from $(p|q)$ be? We recall that the parallel operator of CBS demands that each of p and q participate in any transition of $(p|q)$. For each input transition of p we must have an input or discard transition from q. How to enforce this discipline in a late semantics is not at all clear. The reason for this is that the communication protocol is driven by agents' reactions to *values* being offered. The agent p and q respond to the value 1 by receiving and discarding respectively, but they respond to the value 2 by both receiving. The problem here is that the late semantics for p want to treat the values 1 and 2 in one block of values. There is a single input move from p, this being

$$p \xrightarrow{\{1,2\}?} (x \in \{1,2\})t,$$

which demands a single reaction from q, but of course q cannot provide a single reaction to this transition. For this reason we consider the multiway rendezvous of CBS to be unsuitable to support a late semantics and do not pursue this issue any further.

Chapter 5

Local Model Checking for Value-Passing Processes

In this chapter we consider the problem of model checking for value-passing processes. Model checking refers to the verification that a system, represented as a point of some model, satisfies a property expressed in a particular property logic. One approach to such verification is to interpret properties as sets of points of the model, calculate the relevant set of points for the property which we are interested in and look to see if our system lies in this set. A major drawback here is that the sets of points which interpret formulae can be very costly to calculate. As we are only interested in whether a particular point lies within the property set we need not calculate the whole set. It is often sufficient to check a localised part of the model. This approach to verification is referred to as *local* model checking, [98], and has been pursued extensively, [98, 59, 103, 107, 14].

The property logic which we are going to use is based upon the modal μ-calculus, [61, 88]. This is a very expressive branching time logic, which contains the usual boolean connectives, modal operators to express the ability to perform actions, and maximal and minimal fixpoint operators to express safety and liveness properties. It was shown in [44] that the finite sub-logic of the modal μ-calculus, without fixpoint operators (typically referred to as Hennessy-Milner Logic or HML), is powerful enough to distinguish non-bisimilar (finite branching) processes. Conversely, modal μ-formulae cannot distinguish bisimilar processes, [99]. Although the modal μ-calculus is a very powerful logic we find it lacking for the specification of properties of value-passing processes. Consider the process $p \Longleftarrow i?x.(x < N \rightarrow o!(x+1).\mathbf{O})$. We might want to express the property of p that, after any value is received on i then p makes an output move and the data which is output is one greater than the data received. In HML we could write the formula $F_v \equiv [i?v]\langle o!(v+1)\rangle \mathbf{tt}$, which specifies exactly that property for a single value, v. In order to specify the property for *all* values we would need to use a, possibly infinite, conjunction $\bigwedge_{v \in Val} F_v$. This is clearly undesirable from a specification point of view. These considerations led Hennessy and Liu to develop a first-order version of HML [43] in which the above property is expressed as

$$[i?]\forall x.\langle o!y\rangle y = x+1.$$

We extend this first-order HML by adding maximal and minimal fixpoint operators to the logic. Now, consider the process $p(0)$ where

$$p \Longleftarrow \lambda x.a!x.p(x+2).$$

An evident feature of this process is that it successively outputs the even numbers on data channel a. How could we express such a property in first-order HML with fixpoints? To express that p successively outputs on channel a would require a modality $\langle a!x\rangle$ and a ν-fixpoint. We would have a formula of the form $\nu X.\langle a!x\rangle B \wedge X$ where B is some predicate which says that x is two greater than *the previous output*. To be able to remember what the previous output actually was requires some form of parameterisation. If we call this parameter z we would have the formula

$$\nu X.\langle a!x\rangle(x = z+2) \wedge X(x/z).$$

We can then instantiate this fixpoint's parameter to have an initial value of 0 to express the desired property. Thus we conceive the first-order μ-calculus as the extension of the first-order HML of [43] with maximal and minimal parameterised fixpoints.

We intend to build a tableaux style proof system to verify whether a process satisfies a first-order μ-formula. One of the notable features of such proof systems is the mechanism used to keep track of unfolding fixpoint formulae. There are two common, equivalent, approaches to this problem: introducing constants which name occurrences of fixpoints and using a global rule of inference to detect previous unwindings [103] and introducing *tags* into fixpoint formulae [107], which remember exactly which points of the model have been visited before. We adopt the latter of the two methods, due to Winskel, so we give a brief outline of this to motivate the generalisation. Suppose we are using a transition system model

$$\begin{tikzcd} p \arrow[r, "a"] & q \arrow[l, bend right, "a"'] \end{tikzcd}$$

and we express the safety property *I can perform an infinite sequence of a actions* using the formula

$$\nu X.\langle a\rangle X.$$

Let us try to establish that p satisfies this property; formally we write

$$\vdash p : \nu X.\langle a\rangle X.$$

We can derive this sequent by unwinding the fixpoint formula, but as we unwind we remember that point of the model we are currently at. This is recorded by writing

$$\vdash p : \langle a\rangle \nu X.[p]\langle a\rangle X$$

where the fixpoint formula is now decorated with a tag, $[p]$. An obvious rule for modalities reduces this sequent to

$$\vdash q : \nu X.[p]\langle a\rangle X.$$

Here we must unfold the fixpoint formula once more because we have not met the point q before, so we reduce our goal to

$$\vdash q : \langle a\rangle \nu X.[p,q]\langle a\rangle X.$$

This time recording the fact that we are checking the formula at point q. Another use of the modality rule will yield an obligation,

$$\vdash p : \nu X.[p,q]\langle a\rangle X,$$

at which point we can terminate as p is in the tag set of the formula. We have found a loop in the transition system, which guarantees the possible infinite behaviour.

The way our work differs from Winskel's is that not only do we consider a more expressive property logic, the underlying models that we will use for value-passing processes will in fact be symbolic graphs. It should be clear that the judgements $\vdash p : F$ for checking a pure process term against a modal μ-formula are insufficient in this setting. Recall that nodes of symbolic graphs correspond roughly to open terms of the process language. Our judgements will need to reflect verification that an open process term satisfies a formula possibly containing free data variables and, as such, will be of the form

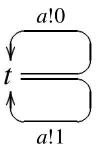
$$B \vdash t : F$$

where B is some boolean condition on data variables, t is the node of a symbolic graph, and F is a formula of first-order μ-calculus with free data variables.

We illustrate how we generalise the proof system a little with the following example. Consider the fixpoint formula

$$A \equiv \nu X.\langle a!x\rangle(x = z mod 2) \wedge X.(z \oplus_2 1/z)$$

where $\oplus_2$ denotes addition modulo 2. We can instantiate A's parameter z to have an initial value of 0 by using the formula $A.(0/z)$. This formula would then state that any process satisfying it could engage in an infinite sequence of output actions on channel a, with the alternating values 0 and 1. We construct the symbolic graph model



and demonstrate that

$$\mathsf{tt} \vdash t : A.(0/z). \tag{5.1}$$

We wish to apply an unwinding rule in order to deal with the fixpoint expression. However, we do not do this immediately as a key rule of the new proof system presented here is a rule which enables us to abstract away from particular instantiations of parameters and deal with fixpoints with arbitrary instantiations of parameters instead. The rule is, roughly, of the form

$$\textit{Subst} \quad \frac{B \vdash t : A}{B[\bar{e}/\bar{z}] \vdash t : A.(\bar{e}/\bar{z})}.$$

We could now choose the boolean B to be the formula $z = 0$ and notice that $\mathsf{tt} \models (z = 0)[0/z]$ to reduce the goal (5.1) to

$$z = 0 \vdash t : A.$$

We can unwind the fixpoint formula A now but we must be careful to note that this judgement is relative to the boolean condition $z = 0$, thus this information must also be recorded in the tag set as we unwind A. Our goal is thus reduced to

$$z = 0 \vdash t : \langle a!x\rangle(x = z mod 2) \wedge A'.(z \oplus_2 1/z)$$

where $A' \equiv \nu X[(z = 0, t)]\langle a!x\rangle(x = z mod 2) \wedge X.(z \oplus_2 1/z)$. Generally, tag sets will contain pairs consisting of a boolean expression and a node of a symbolic graph. We can use a modality rule from [43] and a rule for conjunction to lead to the two judgements

$$z = 0 \vdash t : 0 = z mod 2 \quad \text{and} \quad z = 0 \vdash t : A'.(z \oplus_2 1/z).$$

The former sequent is easily established but the latter requires a further use of the rule Subst. This time we use the boolean expression $z = 1$ which (because $z = 0 \models (z = 1)[z \oplus_2 1/z]$) allows us to reduce our obligation to establishing

$$z = 1 \vdash t : A'.$$

Now the node t does appear in the tag set of A' but we cannot terminate yet because it does so relative to the boolean $z = 0$, not $z = 1$. Thus we need to unwind the formula A' once more, now including $(z = 1, t)$ in the tag set. We continue with similar reasoning and see that we arrive at the sequent

$$z = 1 \vdash t : A''.(z \oplus_2 1/z)$$

where A'' is as A' but with $(z = 1, t)$ also in the tag set. Once again we apply rule Subst using the boolean $z = 0$. Notice that $z = 1 \models (z = 0[z \oplus_2 1/z])$. This leads to the judgement

$$z = 0 \vdash t : A''$$

and this time we can terminate the proof as we do have the pair $(z = 0, t)$ as a tag.

We have successfully established that t satisfies the property $A.(0/z)$ although in a rather unsatisfactory manner. For instance, we had to unwind the formula twice before we could terminate the proof despite there being only one possible node of the model which we could visit. Also, what if the parameters were not updated by addition modulo 2 but by successor on the naturals? The proof would never terminate as we would keep including pairs

$$(z = 0, t), (z = 1, t), (z = 2, t), (z = 3, t), \ldots$$

in the increasing tag set of the formula.

The clear problem with the previous proof is that the initial choice of boolean, $z = 0$, for use with the Subst rule, was not general enough. We resort to checking the fixpoint abstraction first at the single *point* $z = 0$, and then at the point $z = 1$. We refer to this approach as the *pointwise* approach.

Had we chosen the boolean $z = 0 \vee z = 1$ we would see that the proof terminates after one unwinding of the formula. Rather than deal with each *point* individually we adopt an approach which allows us to consider collections of points together. We refer to this as the *abstract* approach. If we were considering successor on the naturals rather than successor modulo 2, we could use the abstract approach and choose the initial boolean to be $even(z) \vee odd(z)$ which would guarantee termination after a single unwinding. The abstract approach proves most useful for tackling formulae which have arbitrary data expressions as parameters to fixpoints because we cannot guarantee a bound on the number of points that will be encountered. However, the pointwise approach does serve a purpose. In a later section of this chapter we will consider a logic whose parameters are such that it will be guaranteed that only finitely many points can be encountered. We benefit by the pointwise approach in that the booleans required for use with the Subst rule are much simpler than those required using the abstract approach.

An essential problem of constructing successful tableaux is the choice of the booleans used with Subst. Proving completeness of our tableaux checker using the abstract approach will involve constructing these booleans to be as general as possible so that each occurrence of a fixpoint formula need only be unwound at most once.

We ought to mention the rôle of tags with respect to least fixpoint formulae. The recurrence of a node in the tag of a greatest fixpoint formula is deemed to be a successful completion of a tableaux. A loop has been found in the model so that infinite behaviour can be guaranteed. In contrast to this, a loop in the model is disastrous for inferring that behaviours are finite. Such properties are specified by least fixpoint formulae and therefore the recurrence of a node in the tag set of a least fixpoint formula is deemed a failure. The tableaux need not continue as it is destined to fail. This property of least fixpoints is reflected by a side-condition on the μ-unwinding rule which forbids further unwinding if a node is found in the tag set. So, whilst tags are useful for inferring satisfaction of maximal fixpoint formulae, they are only useful for inferring failure for minimal fixpoint formulae.

5.1 The specification logic and its interpretation

We give a three-level grammar to describe the logic, $\mathcal{K}\mu$, in Figure 5.1. We assume an ambient data language and let B range over *BoolExp*, c range over *Ch* and X range over a set of *recursion variables* called *RecVar*; note that we generally use lower case $x, y, \ldots$ to denote variables from *Var* and upper case $X, Y, \ldots$ to denote recursion variables. The variable X is bound in the formula $\nu X[\mathcal{A}]F$; write $FV(F)$ for the free recursion variables of F. We use F to range over the

$F ::= B \mid F \vee F \mid F \wedge F \mid \langle\tau\rangle F \mid [\tau]F \mid \langle c!x\rangle F \mid [c!x]F \mid \langle c?\rangle G \mid [c?]G \mid A.(\bar{e}/\bar{x})$

$G ::= \exists x.F \mid \forall x.F$

$A ::= X \mid \nu X[\mathcal{A}]F \mid \mu X[\mathcal{A}]F$

Figure 5.1. Grammar for the logic

main syntactic category of modal formulae while G ranges over quantified formulae. The modal operators $\langle c!x\rangle$ and $[c!x]$ act as binders for the variable x as do the quantifiers $\exists x$ and $\forall x$; write $fv(F)$ for the free data variables of F. This notion is clear for formulae without fixpoints. Finally A ranges over *fixpoint abstractions* but these may only be used to define modal formulae by the construction $A.(\bar{e}/\bar{x})$. This denotes the application of the abstraction A to the substitution $[\bar{e}/\bar{x}]$. This construction binds all of the free data variables of $\bar{x}$ in A, thus we have that

$$fv(A.(\bar{e}/\bar{x})) = fv(e) \cup (fv(A) \setminus \{\bar{x}\})$$

where $fv(e)$ is given by the data-domain and $fv(A)$ is defined by

$$fv(\nu X[\mathcal{A}]F) = fv(\mu X[\mathcal{A}]F) = fv(\mathcal{A}) \cup fv(F) \quad \text{and} \quad fv(X) = \emptyset.$$

The definition of $fv(\mathcal{A})$ is the obvious one. A formula F is called *recursion closed* if $FV(F)$ is empty and is called *data closed* if $fv(F)$ is empty. If A is of the form $\nu X[\mathcal{A}]F$ or $\mu X[\mathcal{A}]F$ then we refer to just the free data variables of F as the *recursion parameters* of A. We will use $\bar{z}$ to denote recursion parameters rather than $\bar{x}$ which denotes an arbitrary vector of data variables. In these formulae we say that the set $\mathcal{A}$ is the tag set for the variable X. This tag set $\mathcal{A}$ will contain pairs (B,t) of boolean expressions and nodes of a symbolic graph. Recall that the occurrence of a node in tag set represents a previous encounter with that node. This means that we must ensure that no information about that previous *state* is lost, for example, nodes in tag sets cannot be α-converted or substituted into. For this reason, when we write $F[v/x]$ for a formula F with v substituted through for free occurrences of x we understand it to mean substitution for the free occurrences of x outside of tag sets. Similarly, we must assume that ! modalities and quantifiers do not bind variables occurring in tag sets. This assumption can be maintained during unfolding of formulae by α-converting to avoid variable capture during substitution. It is important that the free data variables occuring in tag sets correspond exactly to the data variables of the nodes of the symbolic graph being used. We assume that data substitutions behave homomorphically with respect to formula connectives and that they commute with substitutions of formulae for fixpoint abstractions.

The choice to bind the recursion parameters of the formula $A \equiv \nu X[\mathcal{A}]F$ at the stage of application may appear rather unorthodox. For example one might expect that a parameterised (recursion closed) fixpoint formula A is a *closed* formula of type $Val^n \to \mathcal{K}\mu$. This is indeed the case for more standard presentations of parameterised fixpoints where the arity of identifiers is fixed, [67, 66, 79]. In our setting the interpretation of the fixpoint $\nu X[\mathcal{A}]F$ depends on the data variables appearing freely in the tag set, $\mathcal{A}$, as well as the free data variables in F. As we unwind the formula we may need to include additional terms in this tag set, thus the number of variables in $\nu X[\mathcal{A}]F$ may increase dynamically. This means that what we might consider to be the *arity* of an abstraction A could increase through unfolding. Considering the abstractions to be closed, with type $Val^n \to \mathcal{K}\mu$, for some n, would mean that abstractions must be typed dynamically. For this reason it is convenient to consider a recursion closed A simply to be a function from data environments to closed formulae, that is, type $(Var \to Val) \to \mathcal{K}\mu$. Using this type has the benefit that we can give a static type to formulae and also that we can exploit the fact that open formulae may also be construed as functions from data environments to closed formulae in order to give a semantics to fixpoints. In this way, all recursion closed formulae, including abstractions have type

$$
\begin{array}{lcl}
[\![B]\!]\rho\delta & = & \begin{cases} Conc(\mathcal{G}) & \text{If } \delta \models B \\ \emptyset & \text{Otherwise} \end{cases} \\
[\![F \wedge F']\!]\rho & = & [\![F]\!]\rho \cap [\![F']\!]\rho \\
[\![F \vee F']\!]\rho & = & [\![F]\!]\rho \cup [\![F']\!]\rho \\
[\![\langle\tau\rangle F]\!]\rho\delta & = & \left\{ p \mid \exists p' \cdot p \xrightarrow{\tau} p' \text{ and } p' \in [\![F]\!]\rho\delta \right\} \\
[\![[\tau]F]\!]\rho\delta & = & \left\{ p \mid \forall p' \cdot p \xrightarrow{\tau} p' \text{ implies } p' \in [\![F]\!]\rho\delta \right\} \\
[\![\langle c!x\rangle F]\!]\rho\delta & = & \left\{ p \mid \exists p', v \cdot p \xrightarrow{c!v} p' \text{ and } p' \in [\![F[v/x]]\!]\rho\delta \right\} \\
[\![[c!x]F]\!]\rho\delta & = & \left\{ p \mid \forall p', v \cdot p \xrightarrow{c!v} p' \text{ implies } p' \in [\![F[v/x]]\!]\rho\delta \right\} \\
[\![\langle c?\rangle G]\!]\rho\delta & = & \left\{ p \mid \exists (x)p' \cdot p \xrightarrow{c?} (x)p' \text{ and } (x)p' \in [\![G]\!]\rho\delta \right\} \\
[\![[c?]G]\!]\rho\delta & = & \left\{ p \mid \forall (x)p' \cdot p \xrightarrow{c?} (x)p' \text{ implies } (x)p' \in [\![G]\!]\rho\delta \right\} \\
[\![\exists x.F]\!]\rho\delta & = & \{ (y)[t,\delta] \mid \exists v \in Val \cdot ((w)[t[w/y],\delta])v \in [\![F[v/x]]\!]\rho\delta \} \\
[\![\forall x.F]\!]\rho\delta & = & \{ (y)[t,\delta] \mid \forall v \in Val \cdot ((w)[t[w/y],\delta])v \in [\![F[v/x]]\!]\rho\delta \} \\
[\![A.(\bar{e}/\bar{z})]\!]\rho\delta & = & ([\![A]\!]\rho)\delta[\bar{e}/\bar{z}] \\
[\![X]\!]\rho & = & \rho(X) \\
[\![\nu X[\mathcal{A}]F]\!]\rho & = & \nu f.([\![F]\!]\rho[f/X] \cup \lambda\mathcal{A}) \\
[\![\mu X[\mathcal{A}]F]\!]\rho & = & \mu f.([\![F]\!]\rho[f/X] \setminus \lambda\mathcal{A})
\end{array}
$$

where $w = new((y)t, \forall x.F)$, $\lambda\mathcal{A}(\delta) = \{[t,\delta] \mid (B,t) \in \mathcal{A} \text{ and } \delta \models B\}$ and $\cup, \cap$ and $\setminus$ denote pointwise union, intersection and set difference respectively.

Figure 5.2. Interpretation of logic.

$(Var \to Val) \to \mathcal{K}\mu$. This blurring of concepts enables us to give a cleaner presentation of both the semantics of $\mathcal{K}\mu$ and the proof system of Section 3.

In the sequel, for convenience, we will assume that the recursion parameter variables in each fixpoint abstraction are distinct from any variables used in the process terms.

The models of our logic, $\mathcal{K}\mu$, which we shall consider are a subclass of labelled transition systems. In particular, we are interested in exactly those transition systems which are generated as the image of a symbolic graph under concretion. This restriction is only necessary in order to interpret the tags in the fixpoint formulae. The standard transition system models may be used to interpret formula with empty tag sets. Thus, given a symbolic graph, $\mathcal{G}$, we interpret closed formulae as sets of nodes of $Conc(\mathcal{G})$. An open formula will require two environments before it can be interpreted: an environment $\rho : RecVar \to ((Var \to Val) \to \mathcal{P}Conc(\mathcal{G}))$, which provides a meaning for free recursion variables, and an environment $\delta : Var \to Val$, which provides a meaning for free data variables. The meaning of a formula, F, in the environments ρ and δ is written as $[\![F]\!]\rho\delta$. We will let p range over nodes of $Conc(\mathcal{G})$ and denote by $(x)[t,\delta']$ the function $\lambda v.[t,\delta'[v/x]]$. Sometimes it will be convenient to write $t \overset{b,c?}{\longmapsto} (x)t'$ to mean $t \overset{b,c?x}{\longmapsto} t'$. The term $(x)t$ can be thought of as a node of a substitution saturated graph with a specified free variable. Rather than working up to α-equivalence on terms, in order to interpret tag sets cleanly it will be useful to choose a canonical α-equivalence representative for abstractions $(x)t$. To this end, because Var is countable we assume that it is linearly ordered and define a function new which takes an abstraction $(x)t$ and a quantified formula G and returns the least variable which does not occur freely in $(x)t$ and G, except in tag sets. Details of how the interpretations are defined are listed in Figure 5.2, and are as expected. We define a satisfaction relation on open process terms and open formulae using these semantics:

$$t \models_{\rho,B} F \text{ iff } [t,\delta] \in [\![F]\!]\rho\delta \text{ for every } \delta \text{ such that } \delta \models B.$$

When we are using recursion closed formulae this relation is independent of ρ and we simply write $t \models_B F$. For the sub-logic without fixpoint expressions, this satisfaction relation coincides with the one proposed in [43] where it is shown to be characteristic for late bisimulation equivalence. We see that the fixpoints provide no extra distinguishing power over processes.

Proposition 5.1.1 *$t \sim_L^b u$ if and only if for all recursion closed formulae F with empty tag sets,*

$$t \models_b F \text{ iff } u \models_b F$$

Proof Suppose $\delta \models b$, and let p, q denote $[t, \delta]$ and$[u, \delta]$ respectively. The *if* direction is proved in [43] because finite formula suffice to distinguish non-bisimilar processes. We show the converse.

Suppose $p \sim_L u$. We need to show $p \in [\![F]\!]\rho\delta$ iff $q \in [\![F]\!]\rho\delta$. The difficulties arise in the cases of fixpoint formulae. We cannot deal with fixpoints directly but it is sufficient to show that the result holds for their ordinal unwindings. It is well known that $[\![\mu X.F]\!]\rho\delta = \bigcup_\alpha [\![\mu^\alpha X.F]\!]\rho\delta$, [61], where the μ-formulae annotated with an ordinal are interpreted as

$$\begin{array}{lcl} [\![\mu^0 X.F]\!]\rho\delta & = & \emptyset \\ [\![\mu^{\alpha+1} X.F]\!]\rho\delta & = & [\![F[\mu^\alpha X.F/X]]\!]\rho\delta \\ [\![\mu^\gamma X.F]\!]\rho\delta & = & \bigcup_{\alpha<\gamma} [\![\mu^\alpha X.F]\!]\rho\delta \end{array}$$

where γ is a limit ordinal. Similarly, for greatest fixpoints we have a dual interpretation.

If $p \sim_L q$ and $p \in [\![\mu X.F]\!]\rho\delta$ then $p \in [\![\mu^\alpha X.F]\!]\rho\delta$ for some α. If we can prove that $p \in [\![F]\!]\rho\delta$ iff $q \in [\![F]\!]\rho\delta$ for F in the logic featuring only ordinal approximations to fixpoints then this would tell us that $q \in [\![\mu^\alpha X.F]\!]\rho\delta$ so that $q \in [\![\mu X.F]\!]\rho\delta$. A similar argument holds for maximal fixpoint formulae. Therefore we can reduce our obligation to proving the result for a logic with fixpoint approximations.

So, to fulfil our obligation we use well-founded induction over ordinals. Explicitly, we let $F_1 < F_2$ iff F_1 is a proper sub-formula of F_2 or F_1 is $\mu^\beta X.F$ and F_2 is $\mu^\alpha X.F$ with $\beta < \alpha$. This is evidently a well-founded order. We need to consider formulae with free recursion variables. Let θ range over maps from *RecVar* to recursion closed fixpoint formulae. We let $H(F)$ denote the hypothesis, on closed terms, $\forall\delta.(p \in [\![F]\!]\rho\delta \text{ iff } q \in [\![F]\!]\rho\delta)$, and use this to define the hypothesis

$$H^o(F): \ \forall\theta \cdot ((\forall X \in FV(F) \cdot H(\theta X)) \text{ implies } H(\theta F)).$$

It is clear that H^o and H coincide on recursion closed formulae so we choose an arbitrary F and suppose that $H^o(F')$ holds for all $F' < F$. We are to show that $H^o(F)$. Suppose θ is such that $H(\theta X)$ for each $X \in FV(F)$. Consider the possible cases of F.

The base case F is Y for some Y follows by assumption.

For the case $F \equiv A.(\bar{e}/\bar{z})$: Suppose that $p \in [\![\theta A.(\bar{e}/\bar{z})]\!]\rho\delta$. Then $p \in [\![\theta A]\!]\rho\delta[\bar{e}/\bar{z}]$. We know that $H(\theta A)$, by induction, so we have $q \in [\![\theta A]\!]\rho\delta[\bar{e}/\bar{z}]$ whence $q \in [\![\theta A.(\bar{e}/\bar{z})]\!]\rho\delta$. By a symmetric argument we have $H(\theta F)$.

If F is a fixpoint approximation $\mu^\alpha X.F'$ we proceed as follows. If α is 0 then $H(\theta F')$ follows trivially. If α is a limit ordinal then $H(\mu^\beta X.\theta F')$ holds for all $\beta < \alpha$ by induction. We notice that $p \in [\![\mu^\alpha X.\theta F']\!]\rho\delta$ implies $p \in [\![\mu^\beta X.\theta F']\!]\rho\delta$ for some such β. Thus, we obtain $q \in [\![\mu^\beta X.\theta F']\!]\rho\delta$ and $q \in [\![\mu^\alpha X.\theta F']\!]\rho\delta$ accordingly. If α is a successor ordinal $\beta + 1$, say, then we notice that $H(\mu^\beta X.\theta F')$ by induction. We define θ' on $FV(F')$ by

$$\theta'(Y) = \begin{cases} \mu^\beta X.\theta F' & \text{if } Y \equiv X \\ \theta Y & \text{otherwise.} \end{cases}$$

We know that $H(\theta' Y)$ holds for each $Y \in FV(F')$ and also that $F' < F$, so $H^o(F')$. This tells us that $H(\theta' F')$ which is just $H(\theta F'[\mu^\beta X.\theta F'/X])$. It is easy to see then that $H(\mu^\alpha X.\theta F')$.

The case for maximal fixpoint approximations can be done similarly.

The other cases for F are all dealt with, structurally, in [43]. ■

This provides a modal characterisation for late bisimulation equivalence but there is a similar result for early bisimulation if we use a coarser logic. Consider the sub-logic of $\mathcal{K}\mu$ which only uses the input modalities in the following way

$$\langle c?\rangle \exists x.F \text{ and } [c?]\forall x.F.$$

We can prove that this sub-logic characterises early bisimulation in a similar manner to Proposition 5.1.1. We omit the details here as they are very close to [43].

There now follows a technical Lemma which is a generalisation of the so called Reduction Lemma of [107], the essence of the tag set method.

Lemma 5.1.2 *(Reduction Lemma) Let $L = V \to \mathcal{P}T$ be a complete lattice and let $\varphi : L \to L$ be a monotone functional. Let $B \subseteq V$ and write $f \leq_B g$ to mean $\forall v \in B. f(v) \subseteq g(v)$. Then for any $f \in L$,*

$$f \leq_B \nu x.\varphi(x) \textit{ iff } f \leq_B \varphi(\nu x.(\varphi(x) \cup \lambda[B]f))$$

where $\lambda[B]f(v) = f(v)$ if $v \in B$ and is empty otherwise.

Proof Straightforward generalisation of the proof in [107]. ■

It is interesting to note at this point that the corresponding theorem for least fixpoints

$$f \leq_B \mu x.\varphi(x) \text{ iff } f \leq_B \varphi(\mu x.(\varphi(x) \setminus \lambda[B]f))$$

does not hold. To see how this fails consider the following example:

Using the sets $T, V, B = \{a, b\}$ and letting $\lambda\emptyset$ denote the constant empty function we define $\phi(\lambda\emptyset) = f$ where $f(a) = \{a\}, f(b) = \emptyset$ and $\phi(g) = d$ where $d(a) = d(b) = \{a\}$ whenever $g \neq \lambda\emptyset$. It is easy to see that $d = \mu x.\phi(x)$, but notice that $d \not\leq \phi(\mu x.(\phi(x) \setminus d)) = f$.

Lemma 5.1.3 *If $(B', t) \notin \mathcal{A}$ for all B' then*

$$t \models_B \nu X[\mathcal{A}]F \textit{ iff } t \models_B F[\nu X[\mathcal{A}, (B,t)]F/X].$$

Proof Uses Lemma 5.1.2 and simple properties about substitutions. ■

5.2 The proof system

We now present a proof system for verifying whether a formula of the logic holds at a particular point of the model. The system is similar in style to those of [43, 27] in that the proof rules carry side conditions which leave proof obligations of checking implication in some language of boolean conditions and of calculating transitions in a graph. The judgements of the proof system are sequents of the form $B \vdash t : F$ where B is a boolean expression, t is a node of a substitution saturated symbolic graph and F is a recursion closed (but not necessarily data closed) formula of the logic. The proof rules for the system are listed in Figures 5.3 and 5.4. The former imports all of the rules from [43], the latter introduces the rules necessary for the treatment of the fixpoint operators. These new rules are the obvious adaptions of the unfolding rules of [107] to the current setting with the exception of rules *Subst* and η. Note that the assumption that t does not contain any of the recursion parameters $\bar{z}$ is vital for soundness in *Subst*.

The side condition on the μ rule ensures that a term t, say, appearing in a tag set more than once, does so in disjoint boolean worlds. This may seem overly restrictive but it is necessary for soundness. For example we could use the μ-rule to deduce

$$\mathbf{tt} \vdash t : \mu X[(B', t)]\mathbf{tt} \vee X$$

$$Id \quad \frac{}{B \vdash t : B} \qquad\qquad Case \quad \frac{B_1 \vdash t : F, \dots, B_n \vdash t : F}{\bigvee_{1 \le i \le n} B_i \vdash t : F}$$

$$Cons \quad \frac{B_1 \vdash t : F}{B_2 \vdash t : F} \quad (B_2 \models B_1) \qquad\qquad Ex \quad \frac{B \vdash t : F}{\exists x.B \vdash t : F} \quad (x \notin fv(t,F))$$

$$\alpha \quad \frac{B \vdash t' : F'}{B \vdash t : F} \quad (t' \equiv t, F' \equiv F) \qquad\qquad \wedge \quad \frac{B \vdash t : F_1 \quad B \vdash t : F_2}{B \vdash t : F_1 \wedge F_2}$$

$$\vee_L \quad \frac{B \vdash t : F_1}{B \vdash t : F_1 \vee F_2} \qquad\qquad \vee_R \quad \frac{B \vdash t : F_2}{B \vdash t : F_1 \vee F_2}$$

$$\langle\tau\rangle \quad \frac{B \vdash t' : F}{B \wedge b \vdash t : \langle\tau\rangle F} \quad t \overset{b,\tau}{\longmapsto} t'$$

$$[\tau] \quad \frac{B \wedge b_1 \vdash t_1 : F, \dots, B \wedge b_n \vdash t_n : F}{B \vdash t : [\tau]F}$$

$$\text{where } \{(b_1,t_1), \dots, (b_n,t_n)\} = \{(b,t') \mid t \overset{b,\tau}{\longmapsto} t'\}$$

$$\langle c! \rangle \quad \frac{B \vdash t' : F[e/x]}{B \wedge b \vdash t : \langle c!x \rangle F} \quad t \overset{b,c!e}{\longmapsto} t'$$

$$[c!] \quad \frac{B \wedge b_1 \vdash t_1 : F[e_1/x], \dots, B \wedge b_n \vdash t_n : F[e_n/x]}{B \vdash t : [c!x]F}$$

$$\text{where } \{(b_1,t_1,e_1), \dots, (b_n,t_n,e_n)\} = \{(b,t',e) \mid t \overset{b,c!e}{\longmapsto} t'\}$$

$$\langle c? \rangle \quad \frac{B \vdash (y)t' : G}{B \wedge b \vdash t : \langle c? \rangle G} \quad (t \overset{b,c?}{\longmapsto} (y)t')$$

$$[c?] \quad \frac{B \wedge b_1 \vdash (y_1)t_1 : F, \dots, B \wedge b_n \vdash (y_n)t_n : G}{B \vdash t : [c?]G}$$

$$\text{where } \{(b_1,(y_n)t_1), \dots, (b_n,(y_n)t_n)\} = \{(b,(y)t') \mid t \overset{b,c?}{\longmapsto} (y)t'\}$$

$$\forall \quad \frac{B \vdash t : F}{B \vdash (x)t : \forall x.F} \quad (x \notin fv(B)) \qquad\qquad \exists \quad \frac{B \vdash t : F}{B \vdash (x)t : \exists x.F}$$

Figure 5.3. Local model checking rules.

$$\text{Subst}\quad \frac{B \vdash t : A.(\bar{z}/\bar{z})}{B[\bar{e}/\bar{z}] \vdash t : A.(\bar{e}/\bar{z})}\ \bar{z} \notin fv(t) \qquad\qquad \eta\quad \frac{B \vdash t : A}{B \vdash t : A.(\bar{z}/\bar{z})}$$

$$\nu_0\quad \frac{}{B \vdash t : \nu X[\mathcal{A}]F}\ (B,t) \in \mathcal{A} \qquad\qquad \nu_1\quad \frac{B \vdash t : F[\nu X[\mathcal{A} \cup (B,t)]F/X]}{B \vdash t : \nu X[\mathcal{A}]F}$$

$$\mu\quad \frac{B \vdash t : F[\mu X[\mathcal{A} \cup (B,t)]F/X]}{B \vdash t : \mu X[\mathcal{A}]F}\quad \forall B'.(B',t) \in \mathcal{A} \text{ implies } B \wedge B' \models \mathbf{ff}$$

Figure 5.4. Fixpoint rules.

since $\mathbf{tt} \vdash t : \mathbf{tt}$ is trivially derivable. However

$$[\![\mu X[(B',t)]\mathbf{tt} \vee X]\!]\rho\delta = Conc(\mathcal{G}) \setminus \{[t\delta'] \mid \delta' \models B'\}$$

which is different from $Conc(\mathcal{G})$ provided there is some δ such that $\delta \models B'$. This counter-example is mildly pathological in that the false sequent which we derive contains non-empty tags. One might envisage that the side-condition could be replaced with a global condition on proofs which says that if a derivation of a fixpoint sequent can be extended to a derivation of a sequent without tags then the resulting, tag free sequent will be sound. For instance, the example given above could be extended to yield the sequent

$$B' \vdash t : \mu X[\emptyset]\mathbf{tt} \vee X,$$

which happens to be valid.

Theorem 5.2.1 *(Soundness) If $B \vdash t : F$ then $t \models_B F$.*

Proof By induction on the length of the derivation $B \vdash t : F$. The soundness of the crucial rule ν_1 is guaranteed by Lemma 5.1.3 while that of ν_0 follows in a straightforward manner from the interpretation of tagged fixpoints. The soundness of the μ rule is relatively straightforward. We suppose $\delta \models B$ and assume, by induction, that

$$[t,\delta] \in [\![F[\mu X[\mathcal{A} \cup (B,t)]F/X]]\!]\rho\delta.$$

Monotonicity of formulae ensures that

$$[t,\delta] \in [\![F[\mu X[\mathcal{A}]F/X]]\!]\rho\delta$$

and $\delta \not\models B'$ for any $(B',t) \in \mathcal{A}$ implies

$$[t,\delta] \in [\![F[\mu X[\mathcal{A}]F/X]]\!]\rho\delta \setminus \lambda\mathcal{A}(\delta).$$

But this is just to say that

$$[t,\delta] \in [\![\mu X[\mathcal{A}]F]\!]\rho\delta$$

by the semantics of the fixpoint abstraction. We now show the case for the *Subst* rule. Suppose $\delta \models B[\bar{e}/\bar{z}]$. Then we know that $\delta[\bar{e}/\bar{z}] \models B$ and thus, by induction, we have $[t,\delta[\bar{e}/\bar{z}]] \in [\![A.(\bar{z}/\bar{z})]\!]\rho\delta[\bar{e}/\bar{z}]$. Now $\bar{z} \notin fv(t)$ so we have $[t,\delta] \in [\![A.(\bar{z}/\bar{z})]\!]\rho\delta[\bar{e}/\bar{z}]$ which is, by definition, $[t,\delta] \in [\![A.(\bar{e}/\bar{z})]\!]\rho\delta$. Therefore $t \models_{B[\bar{e}/\bar{z}]} A.(\bar{e}/\bar{z})$. ■

Having proved the soundness of our proof system we turn to the question of completeness. It is clear that we will not obtain a general completeness results for arbitrary symbolic graphs. Thus we restrict our attention to finite graphs. Even with this restriction, however, we find that the system is incomplete for the full logic. The problem lies with the least fixpoint formulae. Consider the following example:

Let *Val* be the natural numbers and let the graph $\mathcal{G}$ have two nodes t_1, t_2 with an edge $t_1 \overset{a!x}{\longmapsto} t_2$. The abstraction $\mu X[\emptyset]F$, where F is $(\langle a!y\rangle y = z) \vee X.(z+1/z)$, states *there exists an output on channel a of a value at least as large as z.* We instantiate z at 0 to get the formula which simply reads *there exists an output, on channel a, of some value.* Therefore it should be clear that

$$t_1 \models_{\mathbf{tt}} (\mu X[\emptyset]F).(0/z)$$

We now argue that $\mathbf{tt} \vdash t_1 : (\mu X[\emptyset]F).(0/z)$ is not derivable. First consider the sequence of formulae $F^0 = \mathbf{ff}$ and $F^{n+1} = F[F^n[e/z]/X.(e/z)]$. To see that $t_1 \not\models F^k[0/z]$ for all $k < \omega$ we suppose the contrary, that is $t_1 \models F^k[0/z]$ for some k. Hence for every δ we get that $[t_1,\delta] \in [\![F^k[0/z]]\!]\rho\delta$. Now $[\![F^k[0/z]]\!]\rho\delta = \bigcup_{n<k}\{[t_1,\delta'] \mid \delta'(x) = n\}$. So choose δ_0 such that $\delta_0(x) = k+1$ then by assumption $[t_1,\delta_0] \in \bigcup_{n<k}\{[t_1,\delta'] \mid \delta'(x) = n\}$. This is a contradiction. From this we can conclude that $t_1 \models F^k[0/z]$ for no $k < \omega$.

We now prove that if a term provably satisfies a least fixpoint then it provably satisfies a finite unwinding of this fixpoint. Having done this we easily see, by soundness, that $\mathbf{tt} \vdash t_1 : (\mu X[\emptyset]F).(0/z)$ is not derivable.

Proposition 5.2.2 *If* $B \vdash t : \mu X[\mathcal{A}]F$*, then there exists* $0 \leq k < \omega$ *such that* $B \vdash t : F^k$.

Proof We prove, by induction over depth of proofs, a more general statement: Let F' be an open formula with X as the only possible free recursion variable. Suppose $B \vdash t : F'[\mu X[\mathcal{A}]F/X]$ and let k be the maximum number of occurrences of the μ-rule, using t and X, on any branch of this derivation. Then we have that $B \vdash t : F'[F^{k'}/X]$ for all $k' \geq k$.

This is reasonably straightforward except for the following illustrative cases on which rule was last used:

Case: Conjunction.

$$\wedge \quad \frac{B \vdash t : F_1[\mu X[\mathcal{A}]F/X] \quad B \vdash t : F_2[\mu[\mathcal{A}]F/X]}{B \vdash t : (F_1 \wedge F_2)[\mu X[\mathcal{A}]F/X]}.$$

Suppose without loss of generality that the maximum number of occurrences of the μ-rule appear on the left branch. Then by induction we get $B \vdash t : F_1^{k'}$ for $k' \geq k$. Because we chose k to be maximum across all branches then we also know by induction that $B \vdash t : F_2^{k'}$ for all $k' \geq k_2$ where $k \geq k_2$; in particular, we have this derivation for all $k' \geq k$. Therefore, using rule $\wedge$ we get the result.

Case: Fixpoints. Suppose that the last rule used in the derivation of $B \vdash t : F'[\mu X[\mathcal{A}]F/X]$ was the μ-rule. There are two cases to consider. If F' is simply the formula X then we have

$$\mu \quad \frac{B \vdash t : F[\mu X[\mathcal{A},(B,t)]F/X]}{B \vdash t : \mu X[\mathcal{A}]F}.$$

Induction gives $B \vdash t : F[F^{k'}/X]$ for all $k' \geq k-1$, but this is just $B \vdash t : F^{k'+1}$. Otherwise F' must be of the form $\mu Y[\mathcal{A}']F''$ and we have, writing A for $\mu X[\mathcal{A}]F$,

$$\mu \quad \frac{B \vdash t : (F''[A/X])[\mu Y[\mathcal{A}',(B,t)]F''[A/X]/Y]}{B \vdash t : \mu Y[\mathcal{A}']F''[A/X]}.$$

We can reorder the premise to read $B \vdash t : (F''[\mu Y[\mathcal{A}',(B,t)]F''/Y])[A/X]$ and hence by induction we get

$$B \vdash t : (F''[\mu Y[\mathcal{A}',(B,t)]F''/Y])[F^{k'}/X].$$

Again, by reordering and using the μ-rule on Y we get $B \vdash t : \mu Y.[\mathcal{A}']F''[F^{k'}/X]$ as required.

Case: *Subst.*

$$\begin{array}{rcl}
[\![B]\!]\xi & = & B \\
[\![Q_1 \vee Q_2]\!]\xi & = & [\![Q_1]\!]\xi \vee [\![Q_2]\!]\xi \\
[\![Q_1 \wedge Q_2]\!]\xi & = & [\![Q_1]\!]\xi \wedge [\![Q_2]\!]\xi \\
[\![B \to Q]\!]\xi & = & B \to [\![Q]\!]\xi \\
[\![\exists x.Q]\!]\xi & = & \exists x.[\![Q]\!]\xi \\
[\![\forall x.Q]\!]\xi & = & \forall x.[\![Q]\!]\xi \\
[\![Q_A.(\bar{e}/\bar{z})]\!]\xi & = & ([\![Q_A]\!]\xi)[\bar{e}/\bar{z}] \\
[\![X]\!]\xi & = & \xi(X) \\
[\![\nu X.Q]\!]\xi & = & \bigvee\{b \mid b = [\![Q]\!]\xi[b/X]\}
\end{array}$$

Figure 5.5. Interpretation of first-order fixpoint logic.

We know that F' is a formula $A.(\bar{e}/\bar{z})$ and that B is of the form $B'[\bar{e}/\bar{z}]$. We apply induction to obtain $B' \vdash t : A.(\bar{z}/\bar{z})[F^{k'}/X]$ and then use the *Subst* rule. In the case where F' is $X.(\bar{e}/\bar{z})$ we need a further (easy) induction to show that if $B' \vdash t : F^{k'}$ then $B'[\bar{e}/\bar{z}] \vdash t : F^{k'}[\bar{e}/\bar{z}]$. ■

It is clear from this that the proof system is incomplete for the full logic. Therefore in order to obtain any kind of completeness results we must either augment the proof system or consider sub-logics. We opt for the latter and consider two restricted versions of the logic. In the next section we will prove completeness for the sub-logic obtained by disallowing all μ-formula — a sub-logic of safety properties. In a later section we reintroduce μ-formulae but we restrict the type of expressions that may be passed to them as parameters.

5.3 Completeness for safety properties

In [43] it was shown how it is possible, for finite F, to reduce the statement $t \models F$ to a first-order formula $(t\,\mathrm{sat}\,F)$, thereby reducing the question of a process term satisfying a modal formula down to validity of a boolean expression. We adopt the same approach here. However, to cope with the added complexity of parameterised fixpoints we reduce the statement $t \models F$ to a greatest fixpoint formula over a first-order predicate.

Although first-order logic with fixpoints is a rather unwieldy language with which to reason about data, we benefit by the ability to mechanically describe the most general conditions for which satisfaction will hold; this ability leads us directly to a completeness proof. For example, the conditions $\mathit{even}(z)$ and $\mathit{odd}(z)$, used in the example above would be automatically generated as the solutions of the appropriate fixpoint formulae.

The following grammar describes the logic, $\mathcal{L}\nu$ of first-order predicates with fixpoints.

$$\begin{array}{lcl}
Q & ::= & B \mid Q \vee Q \mid Q \wedge Q \mid B \to Q \mid \exists x.Q \mid \forall x.Q \mid Q_A.(\bar{e}/\bar{z}) \\
Q_A & ::= & X \mid \nu X.Q
\end{array}$$

Here we understand B to be a predicate from *BoolExp*. We present the semantics of this logic by first translating into (possibly open) terms of a first-order logic with infinite disjunction and then interpreting the translations in a completely standard manner. Given a recursion environment $\xi : \mathit{RVar} \to \mathit{BoolExp}$, we translate formulae $[\![Q]\!]\xi$ as described in Figure 5.5 and define $\delta \models [\![Q]\!]\xi$ to be the obvious satisfaction relation. If Q has no free occurrences of recursion variables then $[\![Q]\!]\xi$ is independent of ξ and we simply write $[\![Q]\!]$. In such a case we will write $\delta \models Q$ to mean $\delta \models [\![Q]\!]$ and $Q \vdash t : F$ to mean $[\![Q]\!] \vdash t : F$.

The construction of $(t\,\mathrm{sat}\,F)$ proceeds by structural induction over F. But each time we encounter a new fixpoint abstraction, $\nu X.F'$ say, whilst considering term t, a formula $\nu X_t.\ldots$ is

$$
\begin{array}{lcl}
t\,\mathrm{sat}\,B & = & B \\
t\,\mathrm{sat}\,F_1 \wedge F_2 & = & t\,\mathrm{sat}\,F_1 \wedge t\,\mathrm{sat}\,F_2 \\
t\,\mathrm{sat}\,F_1 \vee F_2 & = & t\,\mathrm{sat}\,F_1 \vee t\,\mathrm{sat}\,F_2 \\
t\,\mathrm{sat}\,\langle\tau\rangle F & = & \displaystyle\bigvee_{t \xrightarrow{b',\tau} t'} b' \wedge (t'\,\mathrm{sat}\,F) \\
t\,\mathrm{sat}\,[\tau]F & = & \displaystyle\bigwedge_{t \xrightarrow{b',\tau} t'} b' \rightarrow (t'\,\mathrm{sat}\,F) \\
t\,\mathrm{sat}\,\langle c!x\rangle F & = & \displaystyle\bigvee_{t \xrightarrow{b',c!e} t'} b' \wedge (t'\,\mathrm{sat}\,F[e/x]) \\
t\,\mathrm{sat}\,[c!x]F & = & \displaystyle\bigwedge_{t \xrightarrow{b',c!e} t'} b' \rightarrow (t'\,\mathrm{sat}\,F[e/x]) \\
t\,\mathrm{sat}\,\langle c?\rangle G & = & \displaystyle\bigvee_{t \xrightarrow{b',c?} (x)t'} b' \wedge ((x)t'\,\mathrm{sat}\,G) \\
t\,\mathrm{sat}\,[c?]G & = & \displaystyle\bigwedge_{t \xrightarrow{b',c?} (x)t'} b' \rightarrow ((x)t'\,\mathrm{sat}\,G) \\
(y)t\,\mathrm{sat}\,\forall x.F & = & \forall w.(t[w/y]\,\mathrm{sat}\,F[w/x]) \;\; w = \mathit{new}((y)t, \forall x.F) \\
(y)t\,\mathrm{sat}\,\exists x.F & = & \exists w.(t[w/y]\,\mathrm{sat}\,F[w/x]) \;\; w = \mathit{new}((y)t, \exists x.F) \\
t\,\mathrm{sat}\,A.(\bar{e}/\bar{z}) & = & (t\,\mathrm{sat}\,A).(\bar{e}/\bar{z}) \\
t\,\mathrm{sat}\,\nu X[\mathcal{A}]F & = & \begin{cases} \lfloor B \rfloor & \text{if } (B,t) \in \mathcal{A} \\ \nu X_t.(t\,\mathrm{sat}\,F[\nu X[\mathcal{A}^+]F/X]) & \text{otherwise} \end{cases}
\end{array}
$$

where $\lfloor B \rfloor = X_t$ if $B \equiv t\,\mathrm{sat}\,\nu X[\mathcal{A}']F$ and $\lfloor B \rfloor = B$ otherwise. Also, $\mathcal{A}^+ = \mathcal{A} \cup (t\,\mathrm{sat}\,\nu X[\mathcal{A}]F, t)$

Figure 5.6. The sat construction.

created. Here the function sat has two arguments, taking the form $t\,\mathrm{sat}\,F$, where t is a node of a symbolic graph and F is a recursion closed formula F. The intention is that $t\,\mathrm{sat}\,F$ describes the weakest condition, B, which guarantees $t \models_B F$.

We list the construction in Figure 5.6. The definition of sat appears to be non-well-founded. There are two reasons for this. Firstly, in the case for fixpoints, we assume that no evaluation of sat formulae occurs inside tag sets. More precisely, consider the tag $(t\,\mathrm{sat}\,\nu X[\mathcal{A}]F, t)$ which is included in the tag set as we unwind a formula against t. While the expression $t\,\mathrm{sat}\,\nu X[\mathcal{A}]F$ is referred to in the context of being in a tag set then we treat it as a symbol only, it is the *name* of a formula rather than the formula itself. Including the expression $t\,\mathrm{sat}\,\nu X[\mathcal{A}]F$ in the tag sets is rather gratuitous as we only really need to remember that t has been encountered before. However, this extra information eases presentation of the following theorems significantly. Secondly, the sat construction will fail to terminate for arbitrary graphs but, if we restrict ourselves to finite graphs, then we see later that the side conditions for defining sat for fixpoints ensure termination. The only problem here though is that $t\,\mathrm{sat}\,F$ is not a well-defined function when F contains two entries for t in a single tag set. Thus we describe a condition on the tag sets in a given formula which guarantees that $t\,\mathrm{sat}\,F$ is in fact well-defined. We call this condition *tag restriction.* A fixpoint formula $\nu X[\mathcal{A}]F$ is ***tag restricted*** if each node t appears at most once in $\mathcal{A}$. More generally, call a formula, F, tag restricted if each of its fixpoint abstractions are tag restricted. Note that a formula with empty tag sets is always tag restricted.

Notice that, in calculating $t\,\mathrm{sat}\,F$, for some node t of a graph $\mathcal{G}$ and some formula F, we need to consider nodes of the form $t''\sigma$, where t'' is reachable from t and σ is a simple substitution used for renaming with fresh variables. We use *reachable* here to mean that there is a sequence of symbolic transitions from t to t''. This requires that $\mathcal{G}$ be a substitution saturated graph. The need for substitution saturation here is, of course, due to the use of α-conversion in the construction of sat. This α-conversion plays two rôles in this construction: firstly, we need to ensure that variables which are re-bound have distinct occurrences in the sat formula. Recall that we do not work up to α-equivalence but with canonical representatives of such classes. Secondly, the connection between the bound variable in the process term and the bound variable which is quantified in the formula is forged merely by converting both to the *same* variable.

If we write $N_{\mathcal{G}}$ for the number of all of the free variables contained in the nodes of $\mathcal{G}$ and N_F for F the number of variables (free and bound) in F, then we only ever have a maximum of $N_t + N_F$ free variables to deal with. This means that, by careful re-use of fresh variables for renaming, we can restrict our data domain to having only a finite set *Var* of variables. Thus, given a finite graph, $\mathcal{G}$, over this restricted data domain we know that the saturated graph $SSat(\mathcal{G})/\sim_{ss}$ is also finite. Hence, we may assume that we are now working with finite, substitution saturated graphs. This is important because nodes of such graphs appear in tag sets. Provided that each term only appears once in each tag set, we have a bound (the size of the graph) on the size of tag sets. We now use this fact to give a well-founded ordering to formulae. We write $F \ll_{\mathcal{G}} F'$ iff F' is not a fixpoint formula and F is a proper sub-formula of F', otherwise F is of the form $F_1[\nu X[\mathcal{A}']F_1/X]$ and F' is $\nu X[\mathcal{A}]F_1$ where $\mathcal{A} \subset \mathcal{A}'$ contain only nodes from $\mathcal{G}$. We aim to show that the transitive closure $\ll^+_{\mathcal{G}}$ of this relation is a well-founded order on tag restricted formulae whenever $\mathcal{G}$ is finite. Given a sequence of vectors $\bar{v}_1, \ldots, \bar{v}_n \ldots$ we will write v^i_n to denote the ith entry of the nth vector.

Proposition 5.3.1 *If $\mathcal{G}$ is finite then $\ll^+_{\mathcal{G}}$ is a well-founded order on tag restricted formulae.*

Proof It is clearly sufficient to prove that $\ll_{\mathcal{G}}$ is well-founded. Suppose otherwise, that there is an infinite decreasing chain

$$\mathcal{C} = \ldots F_n \ll \ldots \ll F_1 \ll F_0$$

of tag-restricted formulae. We aim to show that, eventually, as we progress down the chain, the tag sets of each fixpoint abstraction must contain all possible terms of T, allowing no further unwinding. To do this we consider the partial order $\sqsubset$ which describes the nesting of fixpoint abstractions of F_0. Assume, without loss of generality, that F_0 is a fixpoint abstraction and that all fixpoint sub-formulae of F_0 have uniquely bound recursion variables. Suppose X and Y are distinct recursion variables in F_0 and define $X \sqsubset Y$ if Y is bound within the scope of X. It is clear that $\sqsubset$ is a partial order so we can let

$$\{X_1, X_2, \ldots, X_K\}$$

be the set of all recursion variables in F_0 topologically sorted with respect to $\sqsubset$. That is, if $X_i \sqsubset X_j$ then $i < j$. We will now use A_i to denote an occurrence of a fixpoint sub-formula of some F_n which uses the recursion variable X_i and we will write $A_i \sqsubset A_j$ if $X_i \sqsubset X_j$.

It is clear that, because formulae have finite depth, there must be infinitely many fixpoint unwindings in the chain $\mathcal{C}$. That is, there is an infinite subsequence $\mathcal{C}'$ of $\mathcal{C}$. Such that each F_n of $\mathcal{C}'$ is a fixpoint abstraction A_i. The proof proceeds by defining a sequence of vectors $\{\bar{v}_n\}$ which count the size of the tag sets of occurrences of each A_i in formulae. Loosely speaking, v^i_n will represent $|\mathcal{G}|$ less the size of the tag sets of A_i (denoted $|A_i|_n$) in the nth formula F of $\mathcal{C}'$. More formally, we define

$$\begin{array}{rcl} v^i_0 & = & |\mathcal{G}| - |A_i|_0 \quad \text{for each } i \\ v^i_{n+1} & = & \begin{cases} |\mathcal{G}| - |A_i|_n & \text{if } A_i \text{ is in } F_{n+1} \\ v^i_n & \text{otherwise.} \end{cases} \end{array}$$

It is important to note that the sizes of the tag set for each occurrence of an abstraction A_i in all formulae $F_n \in \mathcal{C}'$ are equal so $|A_i|_n$ is well-defined.

For each n there is always some fixpoint abstraction being unwound at $\mathcal{C}'_n$, so there is always some i, such that $v^i_{n+1} < v^i_n$. Moreover, the tag set of A_i is the only tag set which may increase, thus $v^j_{n+1} \geq v^j_n$ for $j \neq i$. If it is the case that $v^j_{n+1} > v^j_n$ for some j, then we note that this can only occur in the situation where the tag set of A_j *decreases* in size passing from formula F_n to F_{n+1}. We show that the only way this can happen is if $A_i \sqsubset A_j$.

This is slightly tricky to establish. We introduce the idea of a chain with *pointers*. The pointers are designed so that each A_j in each $F_n \in C$ points back to some $A_{j'}$ at $F_{n'}$ in C' (or to some $A_{j'}$ in F_0). We can define the pointers so that all A_i in F_0 point to the smallest sub-formula A_j of F_0 such that $A_j \sqsubset A_i$, with the sub-formula F_0 actually pointing to itself. Subsequently, if F_n has pointers, then F_{n+1} inherits the pointers from F_n and, in the case where F_n is $\nu X[\mathcal{A}]F$ and F_{n+1} is $F[\nu X[\mathcal{A}^+]F/X]$ for some X, we let each occurrence of the newly created sub-formula, $\nu X[\mathcal{A}]F$ in F_{n+1}, point to F_n. An invariant of these pointers is that: A_i in F_n points to the formula A_j at $F_{n'}$ only if $n' = 0$ and $A_j \sqsubset A_i$ or $n' \neq 0$ and $i = j$ and the tag set of A_i is greater than that of A_j. We easily establish this invariant by the definition of pointers at F_0. To see that the invariant is maintained along C we choose some F_n for which it holds and consider why $F_{n+1} \ll F_n$. If F_{n+1} is a sub-formula of F_n then any surviving pointers are inherited and the invariant still holds. Otherwise F_{n+1} is obtained by unfolding, in which case the new pointers satisfy the second part of the invariant.

Resuming our proof, we suppose the chain has such pointers assigned to it. Then we can now see that, in order for the tag set of A_j to *decrease* in passing from F_n to F_{n+1}, the invariant on pointers tells us that every occurrence of A_j in F_{n+1} must point to A_i at F_0 with $A_i \sqsubset A_j$.

This proves that $A_i \sqsubset A_j$ whenever $v^j_{n+1} > v^j_n$. But topological sorting tells us that $i < j$, thus $\bar{v}_{n+1} < \bar{v}_n$ in the lexicographic ordering on vectors because $v^i_{n+1} < v^i_n$. This is true for each n so we have an infinite descending chain of vectors with respect to the lexicographic ordering, which is clearly a contradiction.

■

Now, to see that the definition of $t\,\mathrm{sat}\,F$ is well-founded for t drawn from a finite graph $\mathcal{G}$, and tag-restricted F, we note that

$$F[\nu X[\mathcal{A}^+]F/X] \ll_{\mathcal{G}} \nu X[\mathcal{A}]F$$

and that such an unfolding preserves tag-restrictedness.

As another application of the well-foundedness of $\ll_{\mathcal{G}}$ we show that the characteristic formula construction corresponds to our semantics according to our intentions. The sat construction was designed to characterise satisfaction for formula with empty tag sets. Clearly, as formulae unfold, tag sets become non-empty, however, the booleans in tags are not arbitrary and have very specific forms. In order to help us describe these forms we utilise the notion of a generated pair, (t,F). Generated pairs represent the pairs of nodes of a graph and formulae that one encounter in developing a proof tableaux by using the sat construction. To define this concept fully we use the rewriting relation $\rightarrowtail$ defined in Figure 5.7. So, we say that a pair (t,F) is ***generated*** from (t_0,F_0) (where F_0 has empty tag sets) if $(t,0) \rightarrowtail^* (t,F)$. It should be clear that all formulae F of generated pairs are tag restricted.

Theorem 5.3.2 *For finite $\mathcal{G}$ and empty tag set F_0,*

$$\delta \models t_0\, sat F_0 \text{ iff } [t_0,\delta] \in [\![F_0]\!]\rho\delta.$$

Proof We proceed by well-founded induction with respect to $\ll^+_{\mathcal{G}}$ over formulae of pairs generated from (t_0,F_0). We will actually prove a stronger result that $\delta \models [\![t\,\mathrm{sat}F]\!]\eta$ iff $[t,\delta] \in [\![F]\!]\rho\delta$ where η is some greatest fixpoint environment to be defined below and (t,F) is generated from (t_0,F_0).

We know that (t,F) is generated from (t_0,F_0) so there is a sequence

$$(t_0,F_0) \rightarrowtail^* (t_1,F_1) \rightarrowtail^* \ldots \rightarrowtail^* (t_n,F_n) \rightarrowtail^* (t,F)$$

$$\begin{array}{llll}
(t, F_1 \wedge F_2) & \rightarrowtail & (t, F_1) & \text{and} \quad (t, F_1 \wedge F_2) \rightarrowtail (t, F_2) \\
(t, F_1 \vee F_2) & \rightarrowtail & (t, F_1) & \text{and} \quad (t, F_1 \vee F_2) \rightarrowtail (t, F_2) \\
(t, \langle \tau \rangle F) & \rightarrowtail & (t', F) & \text{for each } t \xmapsto{b,\tau} t' \\
(t, [\tau] F) & \rightarrowtail & (t', F) & \text{for each } t \xmapsto{b,\tau} t' \\
(t, \langle c!x \rangle F & \rightarrowtail & (t', F[e/x]) & \text{for each } t \xmapsto{b,c!e} t' \\
(t, [c!x] F) & \rightarrowtail & (t', F[e/x]) & \text{for each } t \xmapsto{b,c!e} t' \\
(t, \langle c? \rangle G) & \rightarrowtail & ((x)t', G) & \text{for each } t \xmapsto{b,c?x} t' \\
(t, [c?] G) & \rightarrowtail & ((x)t', G) & \text{for each } t \xmapsto{b,c?x} t' \\
((x)t, \forall y.F) & \rightarrowtail & (t[w/x], F[w/y]) & \quad \text{where } w = new(fv((x)t, \forall y.F)) \\
((x)t, \exists y.F) & \rightarrowtail & (t[w/x], F[w/y]) & \quad \text{where } w = new(fv((x)t, \forall y.F)) \\
(t, A.(\bar{e}/\bar{z})) & \rightarrowtail & (t, A) & \\
(t, \nu X[\mathcal{A}]F) & \rightarrowtail & (t, F[\nu X[\mathcal{A}^+]F/X]) \text{ if } t \notin \mathcal{A} &
\end{array}$$

where $\mathcal{A}^+ = \mathcal{A} \cup (t \,\mathrm{sat}\, \nu X[\mathcal{A}]F, t)$.

Figure 5.7. The generated pairs rewriting relation

such that each F_i is of the form $\nu X_i[\mathcal{A}_i]F_i'$ and no fixpoint formulae are encountered between each F_i and F_{i+1}. We define η_1 to be $[[[t_1 \,\mathrm{sat}\, F_1]]/X_{1_{t_1}}]$. Subsequently we define η_{i+1} to be $\eta_i[[[t_{i+1} \,\mathrm{sat}\, F_{i+1}]]\eta_i/X_{i+1_{t_{i+1}}}]$. We will simply write η for η_n and notice that η enjoys the following property:

> if F is of the form $\nu X[\mathcal{A}]F'$ and $(B,t) \in \mathcal{A}$, then B is of the form $t \,\mathrm{sat}\, \nu X[\mathcal{A}']F''$ for some $\mathcal{A}'$ and F'' with $\eta(X_t) = [[B]]\eta$.

We can now proceed with our well-founded induction. We show a couple of the interesting cases.

Case: Application.

$$\begin{array}{lll}
 & \delta \models [[t \,\mathrm{sat}\, A.(\bar{e}/\bar{z})]]\eta & \\
\text{iff} & \delta \models ([[(t \,\mathrm{sat}\, A)]]\eta).(\bar{e}/\bar{z}) & \text{definition of sat} \\
\text{iff} & \delta[\bar{e}/\bar{z}] \models [[t \,\mathrm{sat}\, A]]\eta & \\
\text{iff} & [t, \delta[\bar{e}/\bar{z}]] \in [[A]]\rho\delta[\bar{e}/\bar{z}] & \text{by induction.} \\
\text{iff} & [t, \delta] \in [[A.(\bar{e}/\bar{z})]]\rho\delta. & \text{as } \bar{z} \notin fv(t).
\end{array}$$

Case: ν-fixpoint with $t \in \mathcal{A}$. Then $(B,t) \in \mathcal{A}$ for with $\eta(X_t) = [[B]]\eta$.

$$\begin{array}{lll}
 & \delta \models [[t \,\mathrm{sat}\, \nu X[\mathcal{A}]F]]\eta & \\
\text{iff} & \delta \models [[X_t]]\eta & \text{definition of sat} \\
\text{iff} & \delta \models \eta(X_t) & \\
\text{iff} & \delta \models [[B]]\eta & \\
\text{iff} & [t, \delta] \in [[\nu X[\mathcal{A}]F]]\rho\delta & \text{because } (B,t) \in \mathcal{A}.
\end{array}$$

Case: ν-fixpoint with $t \notin \mathcal{A}$. Let $\mathcal{A}' = \mathcal{A} \cup (t \,\mathrm{sat}\, \nu X.[\mathcal{A}]F, t)$ so that $(t, F[\nu X[\mathcal{A}']F/X])$ is also generated from (t_0, F_0).

$$\begin{array}{lll}
 & \delta \models [[t \,\mathrm{sat}\, \nu X[\mathcal{A}]F]]\eta & \\
\text{iff} & \delta \models [[\nu X_t.t \,\mathrm{sat}\, F[\nu X[\mathcal{A}']F/X]]]\eta & \text{definition of sat} \\
\text{iff} & \delta \models [[t \,\mathrm{sat}\, F[\nu X[\mathcal{A}']F/X]]]\eta & \text{definition of } \eta \\
\text{iff} & [t, \delta] \in [[F[\nu X[\mathcal{A}']F/X]]]\rho\delta & \text{well-founded induction} \\
\text{iff} & [t, \delta] \in [[\nu X[\mathcal{A}]F]]\rho\delta & \text{Lemma 5.1.3}
\end{array}$$

■

Lemma 5.3.3 *For finite $\mathcal{G}$ and pairs (t,F) generated from (t_0,F_0) with η as above:*

$$[\![t\,sat F]\!]\eta \vdash t : F.$$

Proof Similar to the proof in [43] although we use well-founded induction on formulae of generated pairs. The only cases of interest here are application and fixpoint formulae. If t appears in the tag set of the fixpoint formula $\nu X[\mathcal{A}]F$, then rule ν_0 and the definition of η gives the result. Otherwise, by induction we know that

$$[\![t\,\mathrm{sat}F[\nu X[\mathcal{A}']F/X]]\!]\eta \vdash t : F[\nu X[\mathcal{A}']F/X]$$

where $\mathcal{A}' = \mathcal{A} \cup (t\,\mathrm{sat}\,\nu X[\mathcal{A}]F,t)$. But $[\![t\,\mathrm{sat}F[\nu X[\mathcal{A}']F/X]]\!]\eta$ is easily seen to be $[\![t\,\mathrm{sat}\,\nu X[\mathcal{A}]F]\!]\eta$ so by rule ν_1 we have our result.

If F is the formula $A.(\bar{e}/\bar{z})$ then induction tells us that $[\![t\,\mathrm{sat}A]\!]\eta \vdash t : A$. We notice that $[\![t\,\mathrm{sat}A]\!]\eta = [\![(t\,\mathrm{sat}A).(\bar{z}/\bar{z})]\!]\eta$ and apply the rules η and *Subst* to obtain

$$[\![(t\,\mathrm{sat}A)]\!]\eta[\bar{e}/\bar{z}] \vdash t : A.(\bar{e}/\bar{z})$$

from which the result follows. ■

Theorem 5.3.4 *(Completeness) For finite $\mathcal{G}$ and empty tag set F: If $t \models_B F$ then $B \vdash t : F$.*

Proof Suppose $t \models_B F$. Then Theorem 5.3.2 implies that for every $\delta \models B$ we have $\delta \models t\,\mathrm{sat}F$, which is to say $B \models t\,\mathrm{sat}F$. The previous Lemma tells us that $t\,\mathrm{sat}F \vdash t : F$ is derivable so an application of Cons gives $B \vdash t : F$. ■

5.4 Restricted parameters

We now consider an alternative way of restricting the logic in order to obtain a completeness result. Instead of removing least fixpoints completely we limit the usage of the parameters fed to them. We lift the restriction of the previous section so that formulae F may now contain μ-fixpoints. Instead, we say that a formula F has ***restricted parameters*** if for each sub-formula in F of the form $A.(\bar{e}/\bar{z})$ we have that for each $e_i \in \bar{e}$ either

- $e_i = z$ for some z, so that e_i is simply a recursion parameter of F, or
- $var(e_i) \cap \bar{z} = \emptyset$ so e_i contains no recursion parameters at all.

Expressions such as $(z' + 1/z)$ are excluded. For the remainder of this section we assume that F has restricted parameters.

The effect of this restriction is to reduce the role of parameters to that of remembering values (or value expressions) which occur on the arcs of the underlying graphs. When dealing with a finite graph there will be only finitely many value expressions passed as parameters as a formula is unwound. Consider the following example of a formula which demonstrates that the expressive power of our logic is not compromised excessively under parameter restriction: The abstraction A_{fib} is restricted but makes an essential use of parameterisation:

$$A_{fib} = \mu X.[c!x](x = z_1 \wedge [c!y](y = x + z_2 \wedge X.(x+y,y/z_1,z_2))).$$

We see that the formula $A_{fib}.(1,0/z_1,z_2)$ states *I can perform a finite output stream on channel c which follows the Fibonacci sequence.*

The key result of this section then is that, under reasonable conditions, the proof system of Section 5.2 is complete for finite symbolic graphs and restricted parameter formulae.

Theorem 5.4.1 *(Completeness) For all formulae F with empty tag sets, finite $\mathcal{G}$, $fv(B) \subseteq fv(t)$,*

$$t \models_B F \text{ implies } B \vdash t : F.$$

The remainder of this section is devoted to establishing this. We design a new semantics, called the *symbolic semantics*, for the logic to get our completeness result. A symbolic interpretation takes a recursion environment and a boolean expression, rather than a data environment, in order to interpret the free data variables. The boolean represents the set of all data environments which satisfy it. It will be useful to maintain, throughout the completeness proof, a very strict form for these boolean expressions. We assume that they have the form

$$B \wedge (\bar{z} = \bar{e})$$

where B is a boolean expression not containing any recursion parameters and $\bar{e}$ is a finite vector of data expressions also not containing any recursion parameters. For notational convenience we describe such a boolean as follows. Let ε range over substitutions of the form $[\bar{e}/\bar{z}]$ where $\bar{e}$ contains no recursion parameters (note that the identity substitution is the special case of this where the vectors are zero length). Given $\varepsilon = [\bar{e}/\bar{z}]$ and a boolean expression B not containing recursion parameters we write $\widehat{\varepsilon}$ for the boolean expression $\bar{z} = \bar{e}$ and $B\widehat{\varepsilon}$ for $B \wedge \widehat{\varepsilon}$. Using this strict form of boolean environment we present the symbolic interpretation of our logic in Figure 5.8. The first thing that we ought to check is that these symbolic semantics coincide in the correct way with the data environment based semantics. We write $t \models^s_{\rho,B\widehat{\varepsilon}} F$ iff $t \in [\![F]\!]_s\rho B\widehat{\varepsilon}$. By correct we mean that $\models^s$ coincides with $\models$ for formulae without tag sets. The crucial difference between the semantics is in fact the way tags are handled.

Proposition 5.4.2 *If F is a recursion closed formula which has empty tag sets then $t \models_{B\widehat{\varepsilon}} F$ iff $t \models^s_{B\widehat{\varepsilon}} F$.*

Proof Again, it is sufficient to use ordinal unwindings of fixpoints. We prove the result by well-founded induction, with respect to the sub-formula ordering and ordinals, using the hypothesis

$$H(F) : t \models_{B\widehat{\varepsilon}} F \text{ iff } t \models^s_{B\widehat{\varepsilon}} F$$

on recursion closed formulae and

$$H^o(F) : \forall\theta \cdot ((\forall X \in FV(F) \cdot H(\theta X)) \text{ implies } H(\theta F))$$

on open formulae. Choose any F and suppose that $H^o(F')$ holds for any $F' < F$. We are to show that $H^o(F)$. To this end we suppose θ is such that $H(\theta X)$ holds for each $X \in FV(F)$ and consider the possible cases for F.

Case: Atomic Boolean, F is B'.
We know that $\theta F \equiv B'$ and that $t \models_{B\widehat{\varepsilon}} B'$ iff $B\widehat{\varepsilon} \models B'$ iff $t \models^s_{B\widehat{\varepsilon}} B'$. Thus $H^o(B')$ holds easily.

Case: Recursion variable. Holds by assumption.

Case: Conjunction. Simple.

Case: Disjunction. F is $F_1 \vee F_2$.
We can assume that $H(F_i)$ holds for $i = 1,2$. The *if* direction is straightforward. $t \models^s_{B\widehat{\varepsilon}} \theta F_1 \vee \theta F_2$ implies that there exists two booleans B_1, B_2 such that $B\widehat{\varepsilon} \models B_1 \vee B_2$ with $t \models^s_{B_i\widehat{\varepsilon}} \theta F_i$, for $i = 1,2$. By induction we get $t \models_{B_i\widehat{\varepsilon}} \theta F_i$ for $i = 1,2$ so whenever $\delta \models B\widehat{\varepsilon}$ we know that $\delta \models B_i$ for $i = 1$ or $i = 2$, in either case $[t,\delta] \in [\![\theta(F_1 \vee F_2)]\!]\rho\delta$. Hence $t \models_{B\widehat{\varepsilon}} \theta(F_1 \vee F_2)$.

$$
\begin{array}{lcl}
[\![B']\!]_s\rho B\widehat{\varepsilon} & = & \left\{ \begin{array}{ll} \mathcal{G} & \text{If } B\widehat{\varepsilon} \models B' \\ \emptyset & \text{Otherwise} \end{array} \right. \\
[\![F \wedge F']\!]_s\rho B\widehat{\varepsilon} & = & [\![F]\!]_s\rho B\widehat{\varepsilon} \cap [\![F']\!]_s\rho B\widehat{\varepsilon} \\
[\![F \vee F']\!]_s\rho B\widehat{\varepsilon} & = & \bigcup \{ [\![F]\!]_s\rho B_1\widehat{\varepsilon} \cap [\![F']\!]_s\rho B_2\widehat{\varepsilon} \mid B\widehat{\varepsilon} \models B_1 \vee B_2 \} \\
[\![\langle\tau\rangle F]\!]_s\rho B\widehat{\varepsilon} & = & \left\{ \begin{array}{r} t \mid \exists\{c_i\}_I \cdot B\widehat{\varepsilon} \models \bigvee_I c_i, \forall i.\exists t \overset{b_i,\tau}{\longmapsto} t'_i \text{ with } c_i \models b_i \\ \text{and } t'_i \in [\![F]\!]_s\rho(B \wedge c_i)\widehat{\varepsilon} \end{array} \right\} \\
[\![[\tau]F]\!]_s\rho B\widehat{\varepsilon} & = & \left\{ t \mid \forall t \overset{b',\tau}{\longmapsto} t' \text{ implies } t' \in [\![F]\!]_s\rho(B \wedge b')\widehat{\varepsilon} \right\} \\
[\![\langle c!x\rangle F]\!]_s\rho B\widehat{\varepsilon} & = & \left\{ \begin{array}{r} t \mid \exists\{c_i\}_I . B\widehat{\varepsilon} \models \bigvee_I c_i \cdot \forall i.\exists t \overset{b_i,c!e_i}{\longmapsto} t'_i \text{ with } c_i \models b_i \\ \text{and } t'_i \in [\![F[e_i/x]]\!]_s\rho(B \wedge c_i)\widehat{\varepsilon} \end{array} \right\} \\
[\![[c!x]F]\!]_s\rho B\widehat{\varepsilon} & = & \left\{ t \mid \forall t \overset{b',c!e}{\longmapsto} t' \text{ implies } t' \in [\![F[e/x]]\!]_s\rho(B \wedge b')\widehat{\varepsilon} \right\} \\
[\![\langle c?\rangle G]\!]_s\rho B\widehat{\varepsilon} & = & \left\{ \begin{array}{r} t \mid \exists\{c_i\}_I . B\widehat{\varepsilon} \models \bigvee_I c_i \cdot \forall i.\exists t \overset{b_i,c?}{\longmapsto} (y_i)t'_i \text{ with } c_i \models b_i \\ \text{and } (y_i)t'_i \in [\![G]\!]_s\rho(B \wedge c_i)\widehat{\varepsilon} \end{array} \right\} \\
[\![[c?]G]\!]_s\rho B\widehat{\varepsilon} & = & \left\{ t \mid \forall t \overset{b',c?}{\longmapsto} (y)t' \text{ implies } (y)t' \in [\![G]\!]_s\rho(B \wedge b')\widehat{\varepsilon} \right\} \\
[\![\exists x.F]\!]_s\rho B\widehat{\varepsilon} & = & \left\{ \begin{array}{r} (y)t \mid \exists b(w), w = \mathit{new}((y)t, \exists x.F) \cdot B\widehat{\varepsilon} \models \exists w.b(w) \\ \text{and } t[w/y] \in [\![F[w/x]]\!]_s\rho(B \wedge b(w))\widehat{\varepsilon} \end{array} \right\} \\
[\![\forall x.F]\!]_s\rho B\widehat{\varepsilon} & = & \{ (y)t \mid \exists w = \mathit{new}((y)t, \forall x.F) \cdot t[w/y] \in [\![F[w/x]]\!]_s\rho B\widehat{\varepsilon} \} \\
[\![A.(\tilde{e}/\tilde{z})]\!]_s\rho B\widehat{\varepsilon} & = & ([\![A]\!]_s\rho)B[\widehat{\varepsilon(\tilde{e})/\tilde{z}}] \\
[\![X]\!]_s\rho & = & \rho(X) \\
[\![\mu X[\mathcal{A}]F]\!]_s\rho & = & \mu f.([\![F]\!]_s\rho[f/X] \setminus \lambda\mathcal{A}) \\
[\![\nu X[\mathcal{A}]F]\!]_s\rho & = & \nu f.([\![F]\!]_s\rho[f/X] \cup \lambda\mathcal{A})
\end{array}
$$

where $\lambda\mathcal{A}(b) = \{t \mid (b',t) \in \mathcal{A} \text{ and } b \models b'\}$ and $\setminus$ and $\cup$ denote pointwise set difference and union respectively.

Figure 5.8. Symbolic interpretations of formulae

Conversely, suppose that $t \models_{B\widehat{\varepsilon}} \theta(F_1 \vee F_2)$. Let (for $i = 1,2$) B_i be defined by $\delta \models B_i$ iff $\delta \models B\widehat{\varepsilon}$ and $[t,\delta] \in [\![\theta F_i]\!]\rho\delta$. Then $B\widehat{\varepsilon} \models B_1 \vee B_2$ by hypothesis and induction gives $t \models^s_{B_i\widehat{\varepsilon}} \theta F_i$ for $i = 1,2$. Thus $t \models^s_{B\widehat{\varepsilon}} \theta(F_1 \vee F_2)$.

Case:$\langle\alpha\rangle$. We show the case where α is τ.

Suppose $t \models_{B\widehat{\varepsilon}} \langle\tau\rangle\theta F'$, so that $[t,\delta] \in [\![\langle\tau\rangle\theta F']\!]\rho\delta$ whenever $\delta \models B\widehat{\varepsilon}$. Consider the set

$$\{(b_i,t_i')\}_I = \left\{(b',t') \mid t \overset{b',\tau}{\longmapsto} t'\right\}$$

of all symbolic τ transitions from t. We know that whenever $\delta \models B\widehat{\varepsilon}$ there exists an $i \in I$ such that $\delta \models b_i$ and $[t_i',\delta] \in [\![\theta F']\!]\rho\delta$. Define c_i so that $\delta' \models c_i$ iff $\delta' \models (B \wedge b_i)\widehat{\varepsilon}$ and $[t_i',\delta'] \in [\![\theta F']\!]\rho\delta$. We then have $c_i \models b_i$, $B\widehat{\varepsilon} \models \bigvee_I c_i$ and, by induction, $t_i \in [\![\theta F']\!]_s\rho(B \wedge c_i)\widehat{\varepsilon}$. This tells us that $t \models^s_{B\widehat{\varepsilon}} \langle\tau\rangle\theta F'$.

Conversely suppose that $t \in [\![\langle\tau\rangle\theta F']\!]_s\rho B\widehat{\varepsilon}$. By definition we know that there exists $\{c_i\}_I$ such that $B\widehat{\varepsilon} \models \bigvee_I c_i$ and for each $i \in I$ there is a $t \overset{b_i,\tau}{\longmapsto} t'$ such that $c_i \models b_i$ and $t' \in [\![\theta F']\!]_s\rho(B \wedge c_i)\widehat{\varepsilon}$. Let $\delta \models B\widehat{\varepsilon}$, then $\delta \models c_i$ for some i. Thus $\delta \models b_i$ so we know $[t,\delta] \xrightarrow{\tau} [t',\delta]$, and, by induction, $[t',\delta] \in [\![\theta F']\!]\rho\delta$. Therefore $[t,\delta] \in [\![\langle\tau\rangle\theta F']\!]\rho\delta$.

Case: $[\alpha]$. We show the case where α is $c!x$.

Suppose $[t,\delta] \in [\![[c!x]\theta F']\!]\rho\delta$ whenever $\delta \models B\widehat{\varepsilon}$. Now suppose that $t \overset{b',c!e}{\longmapsto} t'$ for some b',t' and that $\delta \models B\widehat{\varepsilon} \wedge b'$. We know that $[t,\delta] \xrightarrow{c!v} [t',\delta]$ for $v = [\![e]\!]\delta$. This means that $[t',\delta] \in [\![(\theta F')[e/x]]\!]\rho\delta$ whenever $\delta \models (B \wedge b')\widehat{\varepsilon}$. Notice that $(\theta F')[e/x] \equiv \theta(F'[e/x])$ as θ substitutes fixpoint abstractions for variables. Thus, by induction, we have $t' \in [\![\theta F'[e/x]]\!]_s\rho(B \wedge b')\widehat{\varepsilon}$. This tells us that $t \in [\![[c!x]\theta F']\!]_s\rho B\widehat{\varepsilon}$.

Conversely, suppose that $t \in [\![[c!x]\theta F']\!]_s\rho B\widehat{\varepsilon}$. We know that $t' \in [\![\theta F'[e/x]]\!]_s\rho(B \wedge b')\widehat{\varepsilon}$ whenever $t \overset{b',c!e}{\longmapsto} t'$. Induction tells us that

$$[t',\delta] \in [\![\theta F'[e/x]]\!]\rho\delta \quad \text{whenever} \quad \delta \models (B \wedge b')\widehat{\varepsilon} \text{ and } t \overset{b',c!e}{\longmapsto} t' \tag{5.2}$$

Now, let $\delta \models B\widehat{\varepsilon}$ and suppose that $[t,\delta] \xrightarrow{c!v} p$. We know that there must exist t',b' and e such that $t \overset{b',c!e}{\longmapsto} t'$ where $\delta \models b'$, $v = [\![e]\!]\delta$ and $p \equiv [t',\delta]$. By (5.2) we know it must be the case that $[t',\delta] \in [\![\theta F'[e/x]]\!]\rho\delta$, whence $[t,\delta] \in [\![[c!x]\theta F']\!]\rho\delta$.

Case: Quantifiers. We show $\exists$, the other case is similar.

Suppose that $(y)[t,\delta] \in [\![\exists x.\theta F']\!]\rho\delta$ whenever $\delta \models B\widehat{\varepsilon}$. This means that for each $\delta \models B\widehat{\varepsilon}$ there is a v_δ such that $[t,\delta[v_\delta/y]] \in [\![\theta F'[v_\delta/x]]\!]\rho\delta$. Note that this v can be chosen so that it depends only on the free variables of $(y)t$ and $\exists x.\theta F'$. We can choose a fresh variable, w and define $b(w)$ abstractly as $\delta \models b(w)$ iff $\delta \models B\widehat{\varepsilon}$ and $\delta(w) = v_\delta$. We immediately notice that $B\widehat{\varepsilon} \models \exists w.b(w)$ because $\delta \models B\widehat{\varepsilon}$ implies $\delta[v_\delta/w] \models b(w)$. Now, whenever $\delta \models (B \wedge b(w))\widehat{\varepsilon}$ we have that $\delta[v_\delta/w] = \delta$. Corollary 2.5.2 tells us that $[t[w/y],\delta[v/w]] \sim [t,\delta[v/y]]$, so, by Proposition 5.1.1 we know that $[t[w/y],\delta] \in [\![\theta F'[w/x]]\!]\rho\delta$ and, by induction, $t[w/y] \in [\![\theta F'[w/x]]\!]_s\rho(B \wedge b(w))\widehat{\varepsilon}$. This tells us that $(y)t \in [\![\exists x.\theta F']\!]_s\rho B\widehat{\varepsilon}$.

Conversely, suppose $(y)t \in [\![\exists x.\theta F']\!]_s\rho B\widehat{\varepsilon}$. Then there exists a fresh variable, w and a boolean $b(w)$ such that $\delta \models B\widehat{\varepsilon}$ implies $\delta[v/w] \models b(w)$ for some v with $t[w/y] \in [\![\theta F'[w/y]]\!]_s\rho(B \wedge b(w))\widehat{\varepsilon}$. So, if $\delta \models B\widehat{\varepsilon}$ then there is some v such that $[t[w/y],\delta[v/w]] \in [\![\theta F'[w/x]]\!]\rho\delta[v/w]$, by induction. Similarly, by Corollary 2.5.2 and Proposition 5.1.1 we know that $[t,\delta[v/y]] \in [\![\theta F']\!]\rho\delta[v/x]$. This is to say that $(y)[t,\delta] \in [\![\exists x.\theta F']\!]\rho\delta$.

Case: Application.

Suppose $t \models_{B\widehat{\varepsilon}} \theta A.(\bar{e}/\bar{z})$ so that whenever $\delta \models B\widehat{\varepsilon}$ we know $[t,\delta] \in [\![\theta A]\!]\rho\delta[\bar{e}/\bar{z}]$. Let $\varepsilon' = [\varepsilon(\bar{e})/\bar{z}]$. Now $\delta' \models \widehat{\varepsilon}'$ implies $\delta' = \delta[\bar{e}/\bar{z}]$ for some $\delta \models \varepsilon$. Therefore, by induction, we know $t \models^s_{B\widehat{\varepsilon}'} \theta A$, whence $t \models^s_{B\widehat{\varepsilon}} \theta A.(\bar{e}/\bar{z})$. The converse is similar.

Case: Fixpoint approximations. We show the case F is $\mu^{\alpha}X.F'$.

Suppose $t \models_{B\widehat{\varepsilon}} \mu^{\alpha}X.\theta F'$. If α is 0 then $H(\theta F)$ holds trivially. If α is a limit ordinal then $H(\mu^{\beta}X.F')$ holds for all $\beta < \alpha$. If $t \models_{B\widehat{\varepsilon}} \theta F$ then $t \models_{B\widehat{\varepsilon}} \mu^{\beta}X.\theta F'$ for some such β. This means that $t \models^{s}_{B\widehat{\varepsilon}} \mu^{\beta}X.\theta F'$, whence $t \models^{s}_{B\widehat{\varepsilon}} \mu^{\alpha}X.\theta F'$. Suppose then that α is $\beta + 1$. We know that $H(\mu^{\beta}X.\theta F')$ holds by well-founded induction on ordinals. So define

$$\theta'(Y) = \begin{cases} \mu^{\beta}X.\theta F' & \text{if } Y \equiv X \\ \theta Y & \text{otherwise.} \end{cases}$$

It is easy to see that $H(\theta Y)$ holds for all $Y \in FV(F')$ and, because $F' < F$ implies that $H^{o}(F')$, we know that $H(\theta' F')$. This is just $H(\theta F'[\mu^{\beta}X.\theta F'/X])$, which, by definition, gives $H(\theta F)$. ■

We now see how fruitful the symbolic semantics are by returning to the Reduction Lemma, Lemma 5.1.2, which failed for least fixpoints previously.

Lemma 5.4.3 *(Reduction lemma revisited) Let M be the lattice of monotone functions from the partial order $(\mathcal{B}, \leq)$ to $(\mathcal{P}T, \subseteq)$ and let $\varphi : M \to M$ be monotone. Then for any $t \in T, b \in \mathcal{B}$,*

$$t \in \nu x.\varphi(x)(b) \textit{ iff } t \in \varphi(\nu x.(\varphi(x) \cup \lambda(b,t)))(b)$$

and

$$t \in \mu x.\varphi(x)(b) \textit{ iff } t \in \varphi(\mu x.(\varphi(x) \setminus \lambda(b,t)))(b)$$

where $\lambda(b,t)(b') = \begin{cases} \{t\} & \textit{If } b' \leq b \\ \emptyset & \textit{otherwise.} \end{cases}$

Proof The result for greatest fixpoints is similar to [107]. We consider the least fixpoint. The if direction is easy. For the only if direction we suppose that $t \in \mu x.\varphi(x)(b)$. Standard fixpoint theory [61] tells us that there exists an ordinal α such that $t \in \varphi^{\alpha}(b)$ where $\varphi^{0} = \lambda\emptyset$ (the constant empty function), $\varphi^{n+1} = \varphi(\varphi^{n})$ and $\varphi^{\gamma} = \bigcup_{\beta<\gamma}\varphi^{\beta}$ when γ is a limit ordinal. Let α be the least such ordinal. M is a lattice of monotone functions so for all $\beta < \alpha$ and $b' \leq b$ we have that $t \notin \varphi^{\beta}(b')$ and so $\varphi^{\beta} = \varphi^{\beta} \setminus \lambda(b,t)$. The result now follows from the monotonicity of φ. ■

Lemma 5.4.4 *If $(B',t) \in \mathcal{A}$ implies $B' \not\models B\widehat{\varepsilon}$ then*

(1) $t \models^{s}_{B\widehat{\varepsilon}} \mu X[\mathcal{A}]F$ iff $t \models^{s}_{B\widehat{\varepsilon}} F[\mu X[\mathcal{A} \cup (B\widehat{\varepsilon},t)]F/X]$

(2) $t \models^{s}_{B\widehat{\varepsilon}} \nu X[\mathcal{A}]F$ iff $t \models^{s}_{B\widehat{\varepsilon}} F[\nu X[\mathcal{A} \cup (B\widehat{\varepsilon},t)]F/X]$.

Proof Follows from previous lemma taking T to be $\mathcal{T}(\mathcal{G})$ and $\mathcal{B}$ to be the boolean expressions (up to equivalence) ordered by $\models^{-1}$. ■

The approach to proving completeness is the same as the proof of the previous section. That is we define a characteristic formula $t\,\text{sat}\,F$ which is the solution a fixpoint formula over a first-order language of boolean expressions. We no longer require parameterised fixpoint formulae as we deal with the recursion parameters using the $B\widehat{\varepsilon}$ statements. This requires knowing the ε part of the environment when calculating $(t\,\text{sat}\,F)$. Figure 5.9 shows how this is done. For each variable X, each term t, and each environment ε, we have a new variable $X_{t\varepsilon}$ and create a formula $\nu X_{t\varepsilon}\ldots$. We do not require a similar construction for least fixpoints here because, as we saw in Proposition 5.2.2, any proof involving least fixpoints can be transformed into a proof involving finite unwindings and therefore any boolean information required to do this proof can also be expressed without least fixpoints. Again we note that the tag sets will contain more information than is strictly necessary to define sat; for fixpoints we only need to record the term t and the environment ε in the tag sets but for the sake of a cleaner presentation later on we include the extra syntax.

$$
\begin{array}{lcl}
\varepsilon \triangleright t \,\mathrm{sat}\, B & = & B[\varepsilon(\bar{z})/\bar{z}] \\
\varepsilon \triangleright t \,\mathrm{sat}\, F_1 \wedge F_2 & = & \varepsilon \triangleright t \,\mathrm{sat}\, F_1 \wedge \varepsilon \triangleright t \,\mathrm{sat}\, F_2 \\
\varepsilon \triangleright t \,\mathrm{sat}\, F_1 \vee F_2 & = & \varepsilon \triangleright t \,\mathrm{sat}\, F_1 \vee \varepsilon \triangleright t \,\mathrm{sat}\, F_2 \\
\varepsilon \triangleright t \,\mathrm{sat}\, \langle \tau \rangle F & = & \bigvee\limits_{t \xmapsto{b', \tau} t'} b' \wedge \varepsilon \triangleright t' \,\mathrm{sat}\, F \\
\varepsilon \triangleright t \,\mathrm{sat}\, [\tau] F & = & \bigwedge\limits_{t \xmapsto{b', \tau} t'} b' \rightarrow \varepsilon \triangleright t' \,\mathrm{sat}\, F \\
\varepsilon \triangleright t \,\mathrm{sat}\, \langle c!x \rangle F & = & \bigvee\limits_{t \xmapsto{b', c!e} t'} b' \wedge \varepsilon \triangleright t' \,\mathrm{sat}\, F[e/x] \\
\varepsilon \triangleright t \,\mathrm{sat}\, [c!x] F & = & \bigwedge\limits_{t \xmapsto{b', c!e} t'} b' \rightarrow \varepsilon \triangleright t' \,\mathrm{sat}\, F[e/x] \\
\varepsilon \triangleright t \,\mathrm{sat}\, \langle c? \rangle G & = & \bigvee\limits_{t \xmapsto{b', c?} (x)t'} b' \wedge \varepsilon \triangleright (x)t' \,\mathrm{sat}\, G \\
\varepsilon \triangleright t \,\mathrm{sat}\, [c?] G & = & \bigwedge\limits_{t \xmapsto{b', c?} (x)t'} b' \rightarrow \varepsilon \triangleright (x)t' \,\mathrm{sat}\, G \\
\varepsilon \triangleright (y)t \,\mathrm{sat}\, \forall x.F & = & \forall w.(\varepsilon \triangleright t[w/y] \,\mathrm{sat}\, F[w/x]) \;\; w = new((y)t, \varepsilon, \forall x.F) \\
\varepsilon \triangleright (y)t \,\mathrm{sat}\, \exists x.F & = & \exists w.(\varepsilon \triangleright t[w/y] \,\mathrm{sat}\, F[w/x]) \;\; w = new((y)t, \varepsilon, \exists x.F) \\
\varepsilon \triangleright t \,\mathrm{sat}\, A.(\bar{e}/\bar{z}) & = & [\varepsilon(\bar{e})/\bar{z}] \triangleright t \,\mathrm{sat}\, A \\
\varepsilon \triangleright t \,\mathrm{sat}\, \nu X[\mathcal{A}] F & = & \begin{cases} \lfloor B \rfloor & \text{if } \exists (B\widehat{\varepsilon}', t) \in \mathcal{A} \text{ with } B\widehat{\varepsilon}' \models \widehat{\varepsilon} \\ \nu X_{t\varepsilon}.(\varepsilon \triangleright t \,\mathrm{sat}\, F[\nu X[\mathcal{A}^{+}] F / X]) & \text{otherwise} \end{cases} \\
\varepsilon \triangleright t \,\mathrm{sat}\, \mu X[\mathcal{A}] F & = & \begin{cases} \mathbf{ff} & \text{if } \exists (B\widehat{\varepsilon}', t) \in \mathcal{A} \text{ with } B\widehat{\varepsilon}' \models \widehat{\varepsilon} \\ (\varepsilon \triangleright t \,\mathrm{sat}\, F[\mu X[\mathcal{A}^{+\mu}] F / X]) & \text{otherwise} \end{cases}
\end{array}
$$

where $\mathcal{A}^{+} = \mathcal{A} \cup ((\varepsilon \triangleright t \,\mathrm{sat}\, \nu X[\mathcal{A}] F)\widehat{\varepsilon}, t)$ and $\mathcal{A}^{+\mu} = \mathcal{A} \cup ((\varepsilon \triangleright t \,\mathrm{sat}\, \mu X[\mathcal{A}] F)\widehat{\varepsilon}, t)$.

Figure 5.9. Sat construction for symbolic semantics

$$\begin{array}{rclcl}
DApps(B) & = & DApps(X) & = & \emptyset \\
DApps(F_1 \wedge F_2) & = & DApps(F_1 \vee F_2) & = & DApps(F_1) \cup DApps(F_2) \\
DApps(\langle \alpha \rangle F) & = & DApps([\alpha]F) & = & DApps(F) \\
DApps(\forall x.F) & = & DApps(\exists x.F) & = & DApps(F) \\
DApps(\mu X[\mathcal{A}]F) & = & DApps(\nu X[\mathcal{A}]F) & = & DApps(F) \\
DApps(A.(e/z)) & = & \multicolumn{3}{l}{\begin{cases} DApps(A) & \text{If } e = z \\ DApps(A) \cup \{e\} & \text{Otherwise.} \end{cases}}
\end{array}$$

Figure 5.10. Definition of function *DApps* over formulae.

We define what it means for a formula F to be tag restricted in a similar manner to before; $\nu X[\mathcal{A}]F$ (or $\mu X[\mathcal{A}]F$) is tag restricted if each node appears at most once in $\mathcal{A}$ for each substitution ε. A formula F is tag restricted if all of its abstraction sub-formulae are tag restricted. A term t can now appear more than once in the tag set of a tag restricted formula. However, any given term t along with a substitution ε may appear at most once.

This change will of course affect our ordering $\ll^+$. The relation $\ll$ given in the previous section is also well-founded for formulae of this sub-logic; this depends on the fact that only a finite number of substitutions, ε, are used as we unwind a formula against a finite graph.

Proposition 5.4.5 *If $\mathcal{G}$ is finite then $\ll_G$ is well-founded on parameter and tag restricted formulae.*

Proof We suppose without loss of generality that our fixpoint formulae only use a single recursion parameter, z. Because the new notion of tag restriction allows a term t to appear several times in each tag set, one for each different ε, it is sufficient, in light of Proposition 5.3.1, to check that we only encounter finitely many ε environments as we calculate $(\varepsilon \triangleright t \,\mathrm{sat} F)$. By inspecting the definition of $(\varepsilon \triangleright t \,\mathrm{sat} F)$ we notice that given an $\varepsilon = [e/z]$ then a new ε' is created only at the application stage, that is

$$\varepsilon \triangleright t \,\mathrm{sat} A.(e'/z) = \varepsilon' \triangleright t \,\mathrm{sat} A$$

where $\varepsilon' = [\varepsilon(e')/z]$. Now the restriction on parameters tells us that either

- $\varepsilon' = [e/z]$ when e' is simply the parameter z or
- $\varepsilon' = [e'/z]$ when e' does not contain recursion parameters.

This observation allows us to describe a general form which the ε must satisfy as we calculate $(\varepsilon \triangleright t \,\mathrm{sat} F)$. We need to describe the possible data expressions which e, e' may be.

We define the function *DApps* inductively over the depth of formulae in Figure 5.10 and note that $DApps(F)$ is finite for any formula F. We let

$$Outs(\mathcal{G}) = \left\{ e \in ValExp \mid \exists t, t' \in \mathcal{G}. t \overset{b,c!e}{\longmapsto} t' \text{ for some } b, c \right\}$$

be the collection of the data expressions which appear on the output arcs of $\mathcal{G}$. For finite $\mathcal{G}$ we see that this set is also finite. *Var* is the *finite* set of variables used in $\mathcal{G}$. Let $BV?(F)$ be the variables bound by quantifiers in F and let $BV!(F)$ be the variables bound by $\langle c! \rangle$ and $[c!]$ modalities in F. We may assume that $BV?(F)$ and $BV!(F)$ are disjoint by α-conversion and they are clearly both finite.

All environments ε' used when calculating $\varepsilon \triangleright t \,\mathrm{sat} F$ are described by the general form

$$\left[e[\bar{w}, \bar{e}'/\bar{x}, \bar{y}]/z \right]$$

where $e \in DApps(F)$, $\bar{x} \in BV?(F)$, $\bar{y} \in BV!(F)$, $\bar{w} \in Var$, and $\bar{e}' \in Outs(\mathcal{G})$. All of the above sets are finite hence there can be finitely many different ε. ■

We must modify the definition of a generated pair slightly. Now we will consider triples (ε, t, F) which can be reached using the $\rightarrowtail$ rewriting relation. We adapt this relation slightly by decorating the rules of Figure 5.7 with ε substitutions in the obvious manner so that

$$(\varepsilon, t, A.(\bar{e}/\bar{z})) \rightarrowtail ([\varepsilon(\bar{e})/\bar{z}], t, A),$$

and

$$(\varepsilon, t, \nu X[\mathcal{A}]F) \rightarrowtail (\varepsilon, t, F[\nu X[\mathcal{A}^+]F/X]) \quad \text{if } (B\widehat{\varepsilon}, t) \notin \mathcal{A} \text{ for any } B$$

where $\mathcal{A}^+ = \mathcal{A} \cup ((\varepsilon \triangleright t \,\mathrm{sat}\, \nu X[\mathcal{A}]F)\widehat{\varepsilon}, t)$. There is a similar rule for least fixpoints.

Proposition 5.4.6 *For all recursion closed formulae, F_0, with empty tag sets, finite $\mathcal{G}$*

$$B\widehat{\varepsilon}_0 \models \varepsilon_0 \triangleright t_0 \mathit{sat} F_0 \text{ iff } t_0 \models^s_{B\widehat{\varepsilon}_0} F_0.$$

Proof We prove by well-founded induction, on triples generated from $(\varepsilon_0, t_0, F_0)$, that, with η defined as in Theorem 5.3.2, we have

$$B\widehat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, F]\!]\eta \text{ iff } t \models^s_{B\widehat{\varepsilon}} F.$$

Case: Boolean. If $t \in [\![B']\!]_s \rho B\widehat{\varepsilon}$ then $B\widehat{\varepsilon} \models B'$. Now, $\delta \models B\widehat{\varepsilon}$ implies $\delta = \delta[\varepsilon(\bar{z})/\bar{z}]$, so $\delta[\varepsilon(\bar{z})/\bar{z}] \models B'$. Thus $\delta \models B'[\varepsilon(\bar{z})/\bar{z}]$, which gives $\delta \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, B']\!]\eta$ as required.

Case: Conjunction. Trivial.

Case: Disjunction.
Suppose $t \in [\![F_1 \vee F_2]\!]_s \rho B\widehat{\varepsilon}$ so that $t \in [\![F_1]\!]_s \rho B_1 \widehat{\varepsilon}$ and $t \in [\![F_2]\!]_s \rho B_2 \widehat{\varepsilon}$ for some B_1, B_2 such that $B\widehat{\varepsilon} \models B_1 \vee B_2$. By induction we know that $B_1\widehat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, F_1]\!]\eta$ and $B_2\widehat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, F_2]\!]\eta$. Thus $B\widehat{\varepsilon} \models (B_1 \vee B_2)\widehat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, F_1 \vee \varepsilon \triangleright t \,\mathrm{sat}\, F_2]\!]\eta \equiv [\![\varepsilon \triangleright t \,\mathrm{sat}\, (F_1 \vee F_2)]\!]\eta$. The converse is given immediately by induction.

Case: $\langle \alpha \rangle$. We show the case α is τ.
Suppose $t \in [\![\langle \tau \rangle F]\!]_s \rho B\widehat{\varepsilon}$. Then there exists $\{c_i\}_I$ such that $B\widehat{\varepsilon} \models \bigvee c_i$ and for each $i \in I$ we have $t \overset{b_i, \tau}{\longmapsto} t'_i$ with $c_i \models b_i$ and $t'_i \in [\![F]\!]\rho(B \wedge c_i)\widehat{\varepsilon}$. Induction tells us that $(B \wedge c_i)\widehat{\varepsilon} \models [\![\varepsilon \triangleright t'_i \,\mathrm{sat}\, F]\!]\eta$. So, for each $\delta \models B\widehat{\varepsilon}$ we know that $\delta \models c_i$ for some i. Thus $\delta \models b_i$ and $\delta \models [\![\varepsilon \triangleright t'_i \,\mathrm{sat}\, F]\!]\eta$ also. This tells us that

$$\delta \models [\![\bigvee_{t \overset{b', \tau}{\longmapsto} t'} b' \wedge \varepsilon \triangleright t' \,\mathrm{sat}\, F]\!]\eta,$$

which implies $B\widehat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, \langle \tau \rangle F]\!]\eta$.
Conversely, if $B\widehat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, \langle \tau \rangle F]\!]\eta$ then we simply let c_i be $b_i \wedge [\![\varepsilon \triangleright t'_i \,\mathrm{sat}\, F]\!]\eta$ and we are done.

Case: $[\alpha].F$. We show the case α is $c!x$.
Suppose that $t \in [\![[c!x]F]\!]_s \rho B\widehat{\varepsilon}$. We know that whenever we have a transition $t \overset{b', c!e'}{\longmapsto} t'$ we have $t' \in [\![F[e'/x]]\!]_s \rho(B \wedge b')\widehat{\varepsilon}$. If we assume that $\delta \models B\widehat{\varepsilon}$ and $\delta \models b'$ for some b' such that $t \overset{b', c!e'}{\longmapsto} t'$ then, by induction, we know that $\delta \models [\![\varepsilon \triangleright t' \,\mathrm{sat}\, F[e'/x]]\!]\eta$, thus $B\widehat{\varepsilon} \models [\![b' \rightarrow \varepsilon \triangleright t' \,\mathrm{sat}\, F[e'/x]]\!]\eta$ for all such b'. Whence $B\widehat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, [c!x]F]\!]\eta$.
Conversely, suppose that $B\widehat{\varepsilon} \models [\![\varepsilon \triangleright t \,\mathrm{sat}\, [c!x]F]\!]\eta$. Then $(B \wedge b')\widehat{\varepsilon} \models [\![\varepsilon \triangleright t' \,\mathrm{sat}\, F[e'/x]]\!]\eta$ for each b', e' such that $t \overset{b', c!e'}{\longmapsto} t'$. This implies that $t' \in [\![F[e'/x]]\!]_s \rho(B \wedge b')\widehat{\varepsilon}$ and consequently $t \in [\![[c!x]F]\!]_s \rho B\widehat{\varepsilon}$.

Case: Quantifiers. Simple.

Case: Application.

Firstly, let ε' denote $[\varepsilon(\bar{e})/\bar{z}]$ and suppose $t \in [\![A.(\bar{e}/\bar{z})]\!]_s \rho B\widehat{\varepsilon}$. Then $t \in [\![A]\!]_s \rho B\widehat{\varepsilon'}$ and, by induction, we know $B\widehat{\varepsilon'} \models [\![\varepsilon' \triangleright t\,\mathrm{sat} A]\!]\eta$. Let $\delta \models B\widehat{\varepsilon}$ so that $\delta[\bar{e}/\bar{z}] \models B\widehat{\varepsilon'}$. This gives us that $\delta[\bar{e}/\bar{z}] \models [\![\varepsilon' \triangleright t\,\mathrm{sat} A]\!]\eta$ and, because $\bar{z}$ does not appear free in $[\![\varepsilon' \triangleright t\,\mathrm{sat} A]\!]\eta$, we have $\delta \models [\![\varepsilon' \triangleright t\,\mathrm{sat} A]\!]\eta \equiv [\![\varepsilon \triangleright t\,\mathrm{sat} A.(\bar{e}/\bar{z})]\!]\eta$.

Conversely, $B\widehat{\varepsilon} \models [\![\varepsilon \triangleright A.(\bar{e}/\bar{z})]\!]\eta$ implies that $B\widehat{\varepsilon} \models [\![\varepsilon' \triangleright t\,\mathrm{sat} A]\!]\eta$. Now, because $\bar{z}$ does not appear free in $[\![\varepsilon' \triangleright t\,\mathrm{sat} A]\!]\eta$ we have that $B\widehat{\varepsilon'} \models B \models [\![\varepsilon' \triangleright t\,\mathrm{sat} A]\!]\eta$ and hence, by induction, $t \in [\![A.(\bar{e}/\bar{z})]\!]_s \rho B\widehat{\varepsilon}$.

Case: Fixpoints.

Straightforward. Uses Lemma 5.4.4 and well-founded induction. ■

The next ingredient in the completeness proof is provabilty from the sat assumption. The side-condition used to define sat at least fixpoints is not strict enough to guarantee the side-condition of the μ proof rule. We recall that this proof rule

$$\mu \quad \frac{B \vdash t : F[\mu X[\mathcal{A} \cup (B,t)]F/X]}{B \vdash t : \mu X[\mathcal{A}]F}$$

had a necessary side-condition that whenever (B',t) appeared in the tag set $\mathcal{A}$ then B and B' were disjoint. With the symbolic interpretation of formulas we can now relax this condition so that we now only require that $B' \not\models B$. This is sufficient to allow us to establish

Lemma 5.4.7 *For all finite $\mathcal{G}$ and triples (ε, t, F) generated from $(\varepsilon_0, t_0, F_0)$ with η as above,*

$$[\![(\varepsilon \triangleright t\,sat F)]\!]\eta\widehat{\varepsilon} \vdash t : F.$$

Proof Again we use $\ll$ for well-founded induction on F. For the most part the proof is similar to that in [43] with the following notable differences:

The base case for atomic propositions requires that $B[\varepsilon(\bar{z})/\bar{z}] \wedge \widehat{\varepsilon} \vdash t : B$. This follows from Id and Cons because $B[\varepsilon(\bar{z})/\bar{z}] \wedge \widehat{\varepsilon} \models B$.

The case for application goes as follows: We know by induction that $[\![(\varepsilon' \triangleright t\,\mathrm{sat} A)]\!]\eta\widehat{\varepsilon'} \vdash t : A$ is derivable (where $\varepsilon' = [\varepsilon(\bar{e})/\bar{z}]$). Using rules η and Subst we get $[\![(\varepsilon' \triangleright t\,\mathrm{sat} A)]\!]\eta\widehat{\varepsilon'}[\bar{e}/\bar{z}] \vdash t : A.(\bar{e}/\bar{z})$. Then, as $\widehat{\varepsilon} \models \widehat{\varepsilon'}[\bar{e}/\bar{z}]$, we see, by Cons, that $[\![(\varepsilon \triangleright t\,\mathrm{sat} A.(\bar{e}/\bar{z}))]\!]\eta\widehat{\varepsilon} \vdash t : A.(\bar{e}/\bar{z})$.

For fixpoints we simply use induction and the unfolding rules when t, ε is not in the tag set and use rule ν_0 or an empty Case otherwise. ■

Proof (Theorem 5.4.1, Completeness)

We prove a slightly different statement, that is, $t \models^s_{B\widehat{\varepsilon}} F$ implies $B\widehat{\varepsilon} \vdash t : F$ and witness the theorem, via Proposition 5.4.2, as an instance of this when ε is the identity. Suppose $t \models^s_{B\widehat{\varepsilon}} F$, then by Proposition 5.4.6 we get $B\widehat{\varepsilon} \models \varepsilon \triangleright t\,\mathrm{sat} F$. The previous Lemma provides a derivation of $(\varepsilon \triangleright t\,\mathrm{sat} F)\widehat{\varepsilon} \vdash t : F$ so an application of Cons will complete the proof. ■

5.5 Example

We now present an extended example of a proof that a finite symbolic graph, compiled from a process declaration, satisfies a property expressed in our logic. In particular, the property is expressed using restricted parameters only and uses both minimal and maximal fixpoint formulae. The process that we declare allows an input stream of non-negative integers on channel i and for each input received performs an output of the maximum value received so far on channel o. The process may be rendered as the parallel composition of two components: A process *Main* which determines the greater of its two inputs on i and k and sends the result on another internal channel m, and a process *Split* which splits the input received on channel m by rerouting it on both o and k. The whole process is illustrated in Figure 5.11.

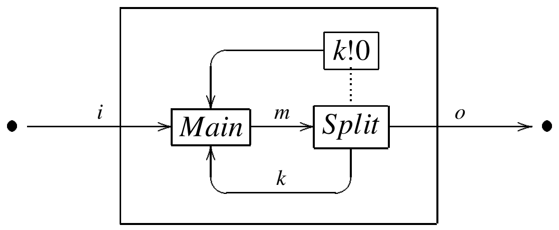


Figure 5.11. Flow diagram for process Max

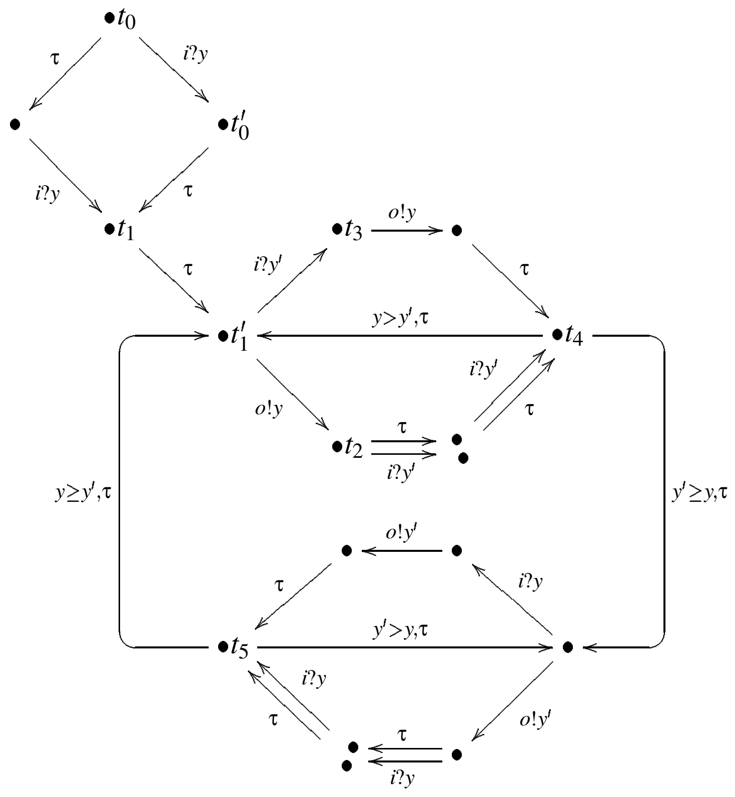


Figure 5.12. Symbolic graph for Max.

Using this description a syntactic version of this process *Max*, is easily obtained.

$$Max \overset{def}{=} (Main \mid k!0.Split)\backslash\{k,m\}$$

where

- $Main \Longleftarrow k?x.i?y.(x > y \rightarrow m!x.Main, m!y.Main)$
  $+ \quad i?y.k?x.(x > y \rightarrow m!x.Main, m!y.Main)$
- $Split \Longleftarrow m?x.o!x.k!x.Split$

Given a syntactic description of a process it is a simple matter to compile it down to a symbolic graph. This treatment can be given to *Max* and we see in Figure 5.12 that the resulting graph is in fact finite. We should point out that at node t_1 in this graph the τ transition leaving this node is guarded by the boolean $y \geq 0$. In light of the fact that y is an non-negative integer we elide this guard. Also, as a companion to this τ transition, there is another τ move with false guard $y < 0$. We prune this branch of the graph for the sake of clarity.

The property that we wish *Max* to satisfy is that for every input on channel i there is an output on channel o of the maximum value received so far. Naturally there are internal actions to be accounted for so we will consider weak modalities, $\langle\langle\alpha\rangle\rangle$ and $[[\alpha]]$. A term will satisfy $\langle\langle\alpha\rangle\rangle F$ if it can do *finitely* many τ transitions followed by an α transition to a term which satisfies F. Because we demand only finitely many τ transitions we use least fixpoints to define these modalities:

$$\langle\langle\alpha\rangle\rangle F \equiv (\mu X.\langle\tau\rangle X.(\bar{z}/\bar{z}) \vee \langle\alpha\rangle F).(\bar{x}/\bar{z})$$

where $\bar{x} = fv(F)$. Similarly for box modalities

$$[[\alpha]]F \equiv (\mu X.[\tau]X.(\bar{z}/\bar{z}) \wedge [\alpha]F).(\bar{x}/\bar{z}).$$

We write the specification as a greatest fixpoint formula,

$$F_{Max} \equiv [[i?]]\forall y.A.(y,0/z,z')$$

where z is a parameter which represents the last value input, z' is a parameter which represents the maximum value received so far and A is defined to be $\nu X.(F_1 \wedge F_2)$. We use two formulae F_1 and F_2 to reflect the fact that, in addition to immediately outputting after an input, the process is able to receive (at most) the next input to be compared before any output transition occurs. These formulae can be written.

$$F_1 \equiv \langle\langle o!x\rangle\rangle[[i?]]\forall y'.F_3 \text{ and } F_2 \equiv [[i?]]\forall y'.\langle\langle o!x\rangle\rangle F_3$$

where

$$F_3 \equiv ([x = z' \wedge z' > z] \vee [x = z \wedge z \geq z']) \wedge X.(y',x/z,z').$$

It is possible to give a proof that $\mathbf{tt} \vdash t_0 : F_{Max}$, however, purely in order to make the proof concise, we use more specific formulae to replace F_1 and F_2. We actually will use

$$F_1 \equiv \langle\langle o!x\rangle\rangle[[i?]]\forall y'.(\langle\tau\rangle)F_3 \vee F_3)$$

and

$$F_2 \equiv [[i?]]\forall y'.\langle o!x\rangle\langle\tau\rangle F_3.$$

These formulae differ from the former two only in their τ modalities.

In the following proof $B \vdash t,t' : F$ will be an abbreviation for the two sequents $B \vdash t : F$ and $B \vdash t' : F$ and B_{Max} will denote the boolean expression $[x = z' \wedge z' > z] \vee [x = z \wedge z \geq z']$ so that $F_3 \equiv B_{Max} \wedge X.(y',x/z,z')$.

The goal $\mathbf{tt} \vdash t_0 : F_{Max}$ follows from rules $\forall$, $[\tau]$, $[i?]$, $\wedge$, μ and Subst if we can establish

$$\mathbf{tt} \vdash t_0', t_1 : A.(y,0/z,z').$$

These can be obtained by using Subst and ν_1 unfolding from

$$\widehat{\varepsilon_0} \vdash t_0', t_1 : (F_1 \wedge F_2)[A_1/X]$$

where $\widehat{\varepsilon_0} \equiv z = y \wedge z' = 0$ and $A_1 = \nu X[\mathcal{A}]F_1 \wedge F_2$ with $\mathcal{A} = (\widehat{\varepsilon_0}, t_0')$ (or, accordingly, $(\widehat{\varepsilon_0}, t_1)$). These judgements can be broken up into the four judgements

$$\widehat{\varepsilon_0} \vdash t_0', t_1 : F_1[A_1/X] \text{ and } \widehat{\varepsilon_0} \vdash t_0', t_1 : F_2[A_1/X]$$

by using the rule $\wedge$. Taking each of these in turn we see that the former pair can be reduced, by using μ unfolding and $\langle\tau\rangle$ rules (twice for the t_0' case) and then a $\langle o!\rangle$ rule to get

$$\widehat{\varepsilon_0} \vdash t_2 : [[i?]]\forall y'(\langle\tau\rangle F_3 \vee F_3)[A_1/X][y/x]. \tag{5.3}$$

Secondly we reduce the latter pair of sequents to

$$\widehat{\varepsilon_0} \vdash t_3 : \langle o!x\rangle\langle\tau\rangle F_3[A_1/X] \tag{5.4}$$

again by using μ-unfoldings and $[\tau]$ rules and a $[i?]$ rule. Now both (5.3) and (5.4) can be reduced to the single judgement $\widehat{\varepsilon_0} \vdash t_4 : F_3[A_1/X][y/x]$ by using the appropriate modality rules. We notice that $\widehat{\varepsilon_0} \models B_{Max}[y/x]$ so, using rules $\wedge$, Cons and Id, our proof obligation becomes $\widehat{\varepsilon_0} \vdash t_4 : A_1.(y', y/z, z')$. We strip the outer application with rule Subst to get

$$\widehat{\varepsilon_1} \vdash t_4 : A_1$$

where $\widehat{\varepsilon_1} \equiv z = y' \wedge z' = y$. This judgement is ready to be ν unfolded to become

$$\widehat{\varepsilon_1} \vdash t_4 : (F_1 \wedge F_2)[A_2/X]$$

where $A_2 = \nu X.[\mathcal{A}, (\widehat{\varepsilon_1}, t_4)]F_1 \wedge F_2$. At this point we do a case analysis on y and y'. This is done by using the Case rule to get the two sequents

$$y > y' \wedge \widehat{\varepsilon_1} \vdash t_4 : (F_1 \wedge F_2)[A_2/X] \tag{5.5}$$

and

$$y' \geq y \wedge \widehat{\varepsilon_1} \vdash t_4 : (F_1 \wedge F_2)[A_2/X]. \tag{5.6}$$

We deal with (5.5) first: this judgement can be divided, using $\wedge$, and then each branch dealt with by using the appropriate modality and μ unfolding rules to obtain the sequent $y > y' \wedge \widehat{\varepsilon_1} \vdash t_4 : F_3[A_2/x][y/x]$. In each case we are using the τ transition going back to node t_1' and following the diamond of transitions to return to t_4. To close this branch of proof we note that $y > y' \wedge \widehat{\varepsilon_1} \models B_{Max}[y/x]$, so we can reduce the statement to $y > y' \wedge \widehat{\varepsilon_1} \vdash t_4 : A_2.(y', y/z, z')$ and, after using rule Subst, that ν_0 is applicable as $(\widehat{\varepsilon_1}, t_4)$ is in the tag set of A_2.

We must now turn our attention to establishing (5.6). Similarly we can divide the judgement into two using $\wedge$ and in both cases follow the appropriate transitions down to the lower part of the graph and through the diamond to get to

$$y' \geq y \wedge \widehat{\varepsilon_1} \vdash t_5 : F_3[A_2/X][y'/x].$$

This judgement is reduced to $y' \geq y \wedge \widehat{\varepsilon_1} \vdash t_5 : A_2.(y, y'/z, z')$ by noting that $y' \geq y \wedge \widehat{\varepsilon_1} \models B_{Max}[y'/x]$ and using the $\wedge$ and Cons rules. We then must apply rule Subst to get

$$\widehat{\varepsilon_2} \vdash t_5 : A_2$$

where $\widehat{\varepsilon_2} \equiv z = y \wedge z = y'$. Now ν_0 is not yet applicable so we must unfold once more to get

$$\widehat{\varepsilon_2} \vdash t_5 : (F_1 \wedge F_2)[A_3/X]$$

where $A_3 = \nu X[\mathcal{A}, (\widehat{\varepsilon_1}, t_4), (\widehat{\varepsilon_2}, t_5)]F_1 \wedge F_2$. The proof now continues in a similar manner to before; we do a case analysis on y and y' to reduce to

$$y' > y \wedge \widehat{\varepsilon_2} \vdash t_5 : (F_1 \wedge F_2)[A_3/X] \ \text{ and } \ y \geq y' \wedge \widehat{\varepsilon_2} \vdash t_5 : (F_1 \wedge F_2)[A_3/X].$$

The left branch follows the modalities back around the lower diamond of the graph and uses the tag $(\widehat{\varepsilon_2}, t_5)$ to close the proof. The right branch uses the τ transition travelling back to the upper part of the graph and uses the tag $(\widehat{\varepsilon_1}, t_4)$ to close the proof and then we are done.

Chapter 6

Unique Fixpoint Induction in Value-Passing CCS

The proof systems of Chapters 2 and 3, and indeed the proof system of [41] upon which these were based suffer from the fact that their use is restricted to the *finite* terms of the languages CBS, and value-passing CCS respectively. A vital feature of process languages is the ability to describe recursive processes. For this reason the proof systems for the sublanguages of finite terms that we have considered are of limited interest. However, they do provide a good basis for the development of more general proof systems for recursively defined agents. In [74] we saw that just three extra inference rules, at least for strong bisimulation, were required for the amelioration of the proof system for finite terms to be able to handle recursively defined agents. The approach used to great effect in [72, 73, 74] is that of *Unique Fixpoint Induction*.

Milner used a fixpoint notation $\mathbf{fix}(X = E)$ to allow recursion in CCS. We prefer to adopt a declarative notation with which a recursive agent can be defined as

$$X \Longleftarrow p$$

where p is a term which may contain the agent constant X. In fact to allow mutual recursion we use a *declaration*, which is a finite set

$$\{X_i \Longleftarrow p_i\}_I,$$

where I is some indexing set with a designated element 1 called the leading element. Any of the constants may appear in each p_i. Suppose for now that we have the singly declared agent, $X \Longleftarrow p$. We can recast Milner's rule of [74] as

$$\text{UFI} \quad \frac{\vdash q = p[q/X]}{\vdash q = X}$$

provided that p is *guarded*, that is any occurrence of the constant X in p must fall within the scope of an action prefix. So that X is guarded in $\alpha.X$ but not in $X + \alpha.X$. At first glance this rule appears to be allowing us to assume that q is provably substitutable for X and then, using this somehow, infer that they are in fact provably equal. Although, in logical terms, this seems a bit like assuming ϕ to prove ϕ, we actually have our assumption about guardedness to make the rule sound. Insisting that p be guarded and substituting q for X ensures that the assumption that $q = X$ cannot be used *immediately*. The UFI rule presented above could not be rendered as follows:

$$\frac{q = X \vdash q = p}{\vdash q = X}$$

because we ought not to have access to the assumption $q = X$ until we have stripped away any prefixes guarding X in p.

The soundness of the proof technique UFI, holds principally due to the coinductive definition of bisimilarity. Consider the two agents

$$X \Longleftarrow \alpha.X \quad \text{and} \quad Y \Longleftarrow \alpha.Y + \alpha.X.$$

We would happily convince ourselves that X and Y are bisimilar by reasoning similar to the reasoning we would use for UFI. We would consider each α-move from Y and match it with a move from X. For example $Y \xrightarrow{\alpha} Y$ is matched by $X \xrightarrow{\alpha} X$ and here, but not before, we assume that X and Y are bisimilar. Formally we would construct $\mathcal{R} = \{(X,Y),(X,X)\}$ as a witnessing bisimulation and then appeal to the coinduction principle which tells us that $\mathcal{R} \subseteq \sim$.

The UFI rule using a full declaration, $\{X_i \Longleftarrow p_i\}_I$, is similar in spirit to the previous rule. We *simultaneously* establish the hypotheses that

$$\vdash q_i = p_i[\bar{q}/\bar{X}]$$

for terms $\{q_i\}_I$ and guarded declarations $\{p_i\}_I$. From this we infer that $q_1 = X_1$.

The purpose of this chapter is to investigate the use of unique fixpoint induction in a value-passing language in order to characterise bisimulation equivalences over a class of recursively defined agents. The particular language we consider is value-passing CCS but we anticipate that the discussion will be applicable to a wider class of process languages, including CBS.

Recently, attempts have been made to generalise this proof technique to process calculi which feature communication of data, [42, 64]. These attempts were reasonably successful in that complete proof systems for strong bisimulation equivalence were found for regular, guarded, terms of both the π-calculus and value-passing CCS. In the latter case, however, completeness was obtained for a class of processes whose parameters were restricted to vectors of names alone. We re-examine this proof system to ascertain why this restriction was necessary and whether we can remove it.

6.1 Abstractions and UFI

The first issue we address in adapting the UFI rule to a value-passing language is the problem of substitution. The hypothesis of the UFI rule utilises the operation $[\bar{q}/\bar{X}]$, substitution of agent constants. What is the correct analogy in a value-passing language? Recall that agent constants for value-passing allow parameterisation, we wish to write terms of the form $X(\bar{v})$. This requires that the constants be considered as functions from values to terms, that is, *abstractions*. Declarations are of the form

$$X \Longleftarrow \lambda\bar{x}.t$$

where all of the free variables of t occur in $\bar{x}$. The substitution operation then will apply to abstractions. We write $t[f/X]$ for the term t with all occurrences of the abstraction identifier X syntactically replaced by f. So it is clear that, in order for this expression to be well-formed, f and X must be of the same arity necessarily. More generally we write $[\bar{f}/\bar{X}]$ as an abbreviation for the operator $[f_i/X_i \mid i \in I]$, the simultaneous substitution of the abstractions f_i for the constants X_i. A precise definition of this operator can be found in [104], as can the following simple facts:

(i) $t[\bar{f}/\bar{X}][\bar{g}/\bar{Y}] \equiv t[\overline{f[\bar{g}/\bar{Y}]},\bar{g}/\bar{X},\bar{Y}]$. Moreover, when the constants appearing in $\bar{f}$ are disjoint with $\bar{Y}$ then $t[\bar{f}/\bar{X}][\bar{g}/\bar{Y}] \equiv t[\bar{f},\bar{g}/\bar{X},\bar{Y}]$

(ii) $t[\bar{e}/\bar{x}][\bar{f}/\bar{X}] \equiv t[\bar{f}/\bar{X}][\bar{e}/\bar{x}]$.

Consider a first attempt at a generalised UFI rule. Recall that the proof systems of [41] and those of Chapters 3 and 4 have sequents of the form $b \rhd t = u$ where t and u are *process terms*, not abstractions. Therefore the UFI rule might look like

$$\frac{\vdash b \rhd u_i = t_i[\bar{f}/\bar{X}]}{\vdash b \rhd u_1 = X_1(\bar{x}_1)}$$

where $f_i \equiv \lambda\bar{x}_i.u_i$ and $\{X_i \Longleftarrow \lambda\bar{x}_i.t_i\}$ is a guarded declaration. This rule, naively stated as above, is unsound. Hennessy and Lin, [42], provide the following example to show this. Let

$$Y \Longleftarrow \lambda x.c!|x|.c?z.Y(z)$$

and

$$X \Longleftarrow \lambda x.c!x.c?z.X(z).$$

The judgement $\vdash x \geq 0 \rhd X(x) = Y(x)$ can be proven using the putative UFI rule above but $X(x)\delta \sim Y(x)\delta$ is clearly untrue for any $\delta \models x \geq 0$. To apply the rule we would have to establish the hypothesis

$$\vdash x \geq 0 \rhd \lambda x.X(x) = c!|x|.c?z.Y(z)[\lambda x.X(x)/Y].$$

This can easily be achieved by β-reducing, unfolding and by virtue of the fact that $x \geq 0 \models x = |x|$.

We consider why this rule fails to be sound. Recall that the unique fixpoint induction described for pure CCS allows one, to *assume* that $q = X$, after doing some work in stripping away prefixes, in order to prove $q = X$. The analogue of this in the previous example would be to assume $x \geq 0 \rhd X(z) = Y(z)$. This is clearly a false assumption, which leads to a false conclusion. A single unwinding of recursive declarations is used for this would-be proof by unique fixpoint induction; the boolean condition $x \geq 0$ is enough to guarantee soundness on the first unwinding only and looks no further ahead. To ensure soundness here we would need to know that $x \geq 0 \models z = |z|$.

The problem with the unsound rule stems from the fact that we are trying to establish properties of process terms like $X(\bar{x})$, whilst the substitution $[\bar{f}/\bar{X}]$ tells us that we are assuming properties of abstractions. This evident disparity can be rectified by considering abstractions in the proof system directly. Instead of using UFI to establish properties of process terms we use it to establish properties of abstractions. This would make the proof rule look like this:

$$\frac{\vdash \mathbf{tt} \rhd f_i = \lambda\bar{x}_i.t_i[\bar{f}/\bar{X}]}{\vdash \mathbf{tt} \rhd f_1 = X_1}.$$

Abstractions are always closed terms so no boolean conditions are necessary. This is a sound rule and directs the development of the proof system towards allowing sequents whose terms may be abstractions.

6.1.1 The proof system

We have already mentioned that the proof systems that we will present are drawn directly from [42], which are based on those of [41] for finite CCS terms. The extra detail carried around by sequents is the name of an underlying declaration. A set of definitions

$$\{X_i \Longleftarrow \lambda\bar{x}_i.t_i \mid i \in I\}$$

is called a ***declaration*** provided that the following conditions hold:

- 1 is a designated element of I,
- the constants appearing in t_i are contained in $\{X_i\}_I$.
- $X_i = X_j$ implies $i = j$.

We say that X_i is ***guarded*** in the declaration D if each occurrence of X_i in any of the terms t_j falls within the scope of an action prefix. That is, X_i is a subexpression of some subexpression $\alpha.t$ of t_j. The whole declaration D is guarded if X_i is guarded in D for each $i \in I$. We will write $\mathcal{T}_D$ for the class of terms which can be built using only the identifiers which have been declared in D.

So, sequents are now of the form

$$\vdash_D b \rhd t = u \quad \text{and} \quad \vdash_D b \rhd f = g$$

where D is a declaration. The declaration given on a sequent is used to identify how to unfold a recursively defined constant. That is, the unfolding rule

$$\frac{}{\vdash_D \mathbf{tt} \rhd X = f} \quad \text{if } X \Longleftarrow f \in D$$

uses D to determine f, the body of X. This could be achieved without having to carry the name D around by simply assuming an underlying declaration, D, and referring to this in side-conditions. However, the declaration is named explicitly on a sequent in order to allow declarations to be extended mid-proof. For example, suppose a proof we are attempting leads to the subgoal

$$\vdash_D b \rhd t = u$$

where t and u are both in $\mathcal{T}_D$. We conceivably might have to introduce a new agent constant X by extending D with an extra definition and then prove

$$\vdash_{D\cup\{X \Longleftarrow \lambda x.t'\}} b \rhd t = X(x) \text{ and } \vdash_{D\cup\{X \Longleftarrow \lambda x.t'\}} b \rhd X(x) = u.$$

Transitivity would only allow us to conclude

$$\vdash_{D\cup\{X \Longleftarrow \lambda x.t'\}} b \rhd t = u$$

but not the desired sequent. We notice though that the terms t and u do not contain the constant X at all and therefore this judgement is valid in the smaller declaration D. We introduce a proof rule which allows us to *shrink* declarations:

$$\frac{\vdash_{D\cup E} b \rhd t = u}{\vdash_D b \rhd t = u} \quad \text{if } t, u \in \mathcal{T}_D$$

and conclude that $\vdash_D b \rhd t = u$. Had we merely assumed an underlying declaration we could not conclude with this sequent.

A full listing of the proof rules of [41], suitably decorated with declarations, appears in Figure 6.1 and the additional rules required for handling constants and abstractions of [42] are given in Figure 6.2. The reader should notice the similarity between the rules of Figures 3.3 and 6.1. The input rule we present here is called L-INPUT to reflect the fact that we will, initially, be characterising late bisimulation; a simple modification to this rule alone suffices to capture early bisimulation. Clearly, the useful properties concerning the manipulation of boolean guards listed in Proposition 3.2.3 also hold for the current proof system and we shall use these liberally.

The set of axioms $\mathcal{A}$ from Chapters 2 and 3 comprises the four laws which state that choice, $+$, forms a commutative, idempotent monoid with **nil** as unit. These four axioms are precisely what is required in the present setting to provide a complete axiomatisation of strong bisimulation. Generally, we will write $\mathcal{B} \vdash_D b \rhd t = u$ to mean that the sequent $\vdash_D b \rhd t = u$ can be derived using the inference rules above and the axioms in $\mathcal{B}$. For readability we will omit the $\mathcal{B}$ whenever it is either irrelevant or clear from context.

As it stands, performing even simple derivations in this proof system is somewhat laborious. However, much of the work involved lies in repetitive applications of β-reductions and λ-eliminations. In an attempt to make the proof system more accessible one might show that common derivation steps can be packaged up into a single derivable rule. For example, the rule

$$SUBST \quad \frac{\vdash_D b \rhd t = u}{\vdash_D b\sigma \rhd t\sigma = u\sigma}$$

is derivable, for any substitution σ. Hennessy and Lin, [42], conceived the following, derivable, rule of inference:

EQUIV $$\frac{}{\vdash_D \mathbf{tt} \rhd t = t} \qquad \frac{\vdash_D b \rhd t = u}{\vdash_D b \rhd u = t} \qquad \frac{\vdash_D b \rhd t = u \quad \vdash_D b \rhd u = v}{\vdash_D b \rhd t = v}$$

AXIOM $$\frac{t = u \in \text{Axioms}}{\vdash_D \mathbf{tt} \rhd t = u}$$

CONG $$\frac{\vdash_D b \rhd t_1 = u_1 \quad \vdash_D b \rhd t_2 = u_2}{\vdash_D b \rhd t_1 + t_2 = u_1 + u_2}$$

α-CONV $$\frac{}{\vdash_D \mathbf{tt} \rhd c?x.t = c?y.t[y/x]} \quad \text{if } y \notin fv(t)$$

L-INPUT $$\frac{\vdash_D b \rhd t = u}{\vdash_D b \rhd c?x.t = c?x.u} \quad \text{if } x \notin fv(b)$$

OUTPUT $$\frac{b \models e = e' \quad \vdash_D b \rhd t = u}{\vdash_D b \rhd c!e.t = c!e'.u}$$

TAU $$\frac{\vdash_D b \rhd t = u}{\vdash_D b \rhd \tau.t = \tau.u}$$

GUARD $$\frac{\vdash_D b \wedge b' \rhd t = u \quad \vdash_D b \wedge \neg b' \rhd \mathbf{nil} = u}{\vdash_D b \rhd b' \rightarrow t = u}$$

CONS $$\frac{\vdash_D b' \rhd t = u}{\vdash_D b \rhd t = u} \quad \text{if } b \models b'$$

CASE $$\frac{\vdash_D b_1 \rhd t = u \ldots \vdash_D b_n \rhd t = u}{\vdash_D \bigvee_{1 \le i \le n} b_i \rhd t = u}$$

Figure 6.1. Inference rules for finite CCS

dec-I $$\frac{\vdash_D b \rhd t = u}{\vdash_{D\cup E} b \rhd t = u}$$

dec-E $$\frac{\vdash_{D\cup E} b \rhd t = u}{\vdash_D b \rhd t = u} \quad \text{if } t, u \in \mathcal{T}_D$$

FIX $$\frac{}{\vdash_D \mathbf{tt} \rhd X = f} \quad \text{if } X \Longleftarrow f \in D$$

UFI $$\frac{\forall i \in I \quad \vdash_D \mathbf{tt} \rhd g_i = f_i[\bar{g}/\bar{X}]}{\vdash_{D\cup E} \mathbf{tt} \rhd g_1 = X_1} \quad \begin{array}{l}\text{where } E = \{X_i \Longleftarrow f_i\}_I \\ \text{is a guarded declaration}\end{array}$$

λ-I $$\frac{\vdash_D b \rhd f(\bar{x}) = g(\bar{x})}{\vdash_D b \rhd f = g} \quad \text{if } \bar{x} \notin fv(b) \text{ and } x_i \neq x_j \text{ for } i \neq j$$

λ-E $$\frac{\vdash_D b \rhd f = g}{\vdash_D b \rhd f(\bar{e}) = g(\bar{e}')} \quad \text{if } b \models \bar{e} = \bar{e}'$$

β $$\frac{}{\vdash_D \mathbf{tt} \rhd (\lambda\bar{x}.t)(\bar{e}) = t[\bar{e}/\bar{x}]}$$

Figure 6.2. New inference rules for constants and abstractions

If $E = \{X_i \Longleftarrow \lambda\bar{x}_i(b_i \to t_i)\}_I$ is a guarded declaration then

$$UFI-O \ \frac{\forall i \in I \quad \vdash_D b_i \rhd f_i(\bar{x}_i) = t_i[\bar{f}/\bar{X}]}{\vdash_{D\cup E} b_1 \rhd f_1(\bar{x}_1) = X_1(\bar{x}_1)}$$

where $f_i \equiv \lambda\bar{x}_i(b_i \to u_i)$ for some terms u_i.

It is this version of the UFI rule that lies at the heart of the completeness proofs of [42]. For this reason we take a closer look at the rule and ask why it is sound.

Firstly, imagine an application of this rule in which we are trying to establish $b_1 \rhd f_1(\bar{x}_1) = X_1(\bar{x}_1)$ by assuming that $b_i \rhd f_i = X_i$ is valid for each i, with the restriction that we only use these assumptions in the scope of an action prefix. Now, suppose we are trying to establish the hypothesis $b_i \rhd f_i(\bar{x}_i) = t_i$ and in doing so, after we remove a prefix from t_i, we reach a subgoal which requires us to show

$$b_i \wedge b' \rhd f_j(\bar{e}) = X_j(\bar{e}) \tag{6.1}$$

for some boolean b'. Hennessy and Lin now observed that we can only soundly use the assumption $b_j \rhd f_j = X_j$ in the boolean world b_j not $b_i \wedge b'$. In order to ensure that we will always be in the world b_j when the assumption is to be used they insisted that the boolean, b_j, guard both the declared identifier X_j and the abstraction f_j. Also, in a similar vein, they found that the assumption $b_j \rhd f_j = X_j$ cannot be used to prove (6.1) unless it is the case that $\bar{e} = \bar{x}_j$. Demanding this leads to using a restricted language where the parameter $\bar{e}$ is a vector of variables alone. This restriction allows a saturation of declarations with all possible permutations of this vector, which is enough to guarantee that $\bar{e} = \bar{x}_j$ will hold.

A valid application of the unique fixpoint induction proof method, to prove that recursive processes p and q are bisimilar, corresponds roughly to finding matching loops in the underlying transition graphs of p and q. In the current value-passing setting we consider *symbolic graphs with assignment* to be the underlying models. Finding matching loops in symbolic graphs with assignments involves not only finding loops with corresponding actions but also ensuring that the data which is output from each graph is equivalent and also that the assignments made during the loop are equivalent. In other words, we need to establish a *loop invariant*, b, which guarantees that the output is equivalent, and we need to show that the invariant is preserved by assignments. For example, consider the processes $X(x)$ and $Y(y)$ where

$$X \Longleftarrow \lambda x.c!x.X(x+1)$$

and

$$Y \Longleftarrow \lambda y.c!(-y).Y(y-1).$$

These definitions describe the same process under the constraint $x = -y$. Thus, a successful application of a more general version of UFI would establish $x = -y$ as the loop invariant and show that $x = -y$ implies $x+1 = -(y-1)$.

Drawing upon our analogy with loop invariants we now see that the restrictions that Hennessy and Lin impose are more than is necessary. We simply require that the invariant(s) $\{b_i\}_I$ be preserved by the substitutions created by recursive calls such as $X_j(\bar{e})$ in (6.1). This amounts to asking that

$$b_i \wedge b' \models b_j[\bar{e}/\bar{x}_j]$$

whenever we encounter a subgoal such as (6.1). To formalize this we syntactically define a function $Calls(X_i)$ on identifiers in D which returns the set of $(b, j, \bar{e})$ such that $X_j(\bar{e})$ appears as a subexpression of t_i guarded by the boolean condition b. This is done by defining the relation $\gg$ by induction on terms, having first ensured that bound names appear uniquely within terms by using α-conversion:

- $X_j(\bar{e}) \overset{\mathbf{tt}}{\gg} (j, \bar{e})$
- $t \overset{b}{\gg} (j, \bar{e})$ implies $\alpha.t \overset{b}{\gg} (j, \bar{e})$ and $t\backslash c \overset{b}{\gg} (j, \bar{e})$
- $t_1 \overset{b}{\gg} (j, \bar{e})$ or $t_2 \overset{b}{\gg} (j, \bar{e})$ implies $t_1 + t_2 \overset{b}{\gg} (j, \bar{e})$ and $t_1|t_2 \overset{b}{\gg} (j, \bar{e})$
- $t \overset{b'}{\gg} (j, \bar{e})$ implies $b \rightarrow t \overset{b \wedge b'}{\gg} (j, \bar{e})$

Given this relation we define $Calls(X_i)$ simply to be the set

$$\left\{(b, j, \bar{e}) \mid t_i \overset{b}{\gg} (j, \bar{e})\right\}.$$

We see how the ideas about loop invariants translate into a new version of the UFI rule:

If $E = \{X_i \Longleftarrow \lambda\bar{x}_i.t_i\}_I$ is a guarded declaration and $f_i \equiv \lambda\bar{x}_i.u_i$ is a collection of abstractions. Then

$$UFI-Inv \ \frac{\vdash_D b_i \rhd f_i(\bar{x}_i) = t_i[\bar{f}/\bar{X}] \quad \forall i \in I}{\vdash_{DUE} b_1 \rhd f_1(\bar{x}_1) = X_1(\bar{x}_1)}$$

provided that $b_i \wedge b \models b_j[\bar{e}/\bar{x}_j]$ whenever $(b, j, \bar{e}) \in Calls(X_i)$.

Intuitively one can see that our version of UFI is sound. Moreover, it gives a tighter analysis of the structural reasoning involved in a proof by unique fixpoint induction for value-passing processes. A testament to its power is the fact that we will now obtain stronger completeness results than those present in [42, 64]. Firstly, though, we show that UFI-Inv is compatible with Hennessy and Lin's proof system by deriving it from UFI. Note that this derivation requires congruence properties of $|$ and $\backslash c$ in the proof system. These congruence properties will be provable in the presence of the expansion laws of Figure 6.3.

Proposition 6.1.1 *UFI-Inv is a derivable rule.*

Proof We actually derive the rule from UFI-O which was itself proven to be derivable in [42]. Suppose

$$\vdash_D b_i \rhd f_i(\bar{x}_i) = t_i[\bar{f}/\bar{X}]$$

for each $i \in I$. Let $f_i' \equiv \lambda \bar{x}_i(b_i \rightarrow f_i(\bar{x}_i))$ and let E' be the declaration constructed from E by replacing each $X_i \Longleftarrow \lambda \bar{x}_i.t_i$ with $X_i' \Longleftarrow \lambda \bar{x}_i(b_i \rightarrow t_i')$ and $t_i' \equiv t_i[\bar{X}'/\bar{X}]$. It is easy to see that

$$\vdash_D b_i \rhd f_i(\bar{x}_i) = f_i'(\bar{x}_i) \tag{6.2}$$

and

$$\vdash_{D \cup E \cup E'} b_i \rhd X_i(\bar{x}_i) = X_i'(\bar{x}_i).$$

By using the side-condition that $b_i \wedge b \models b_j[\bar{e}/\bar{x}_j]$ for each $(b, j, \bar{e}) \in \mathit{Calls}(X_i)$, we use structural induction on the term t_i to prove that

$$\vdash_{D \cup E \cup E'} b_i \rhd t_i[\bar{f}/\bar{X}] = t_i'[\bar{f}'/\bar{X}']. \tag{6.3}$$

Consider the structure of t_i.

Case t_i is $X_j(\bar{e})$. Then $(\mathrm{tt}, j, \bar{e}) \in \mathit{Calls}(X_i)$. We need to show that

$$\vdash_{D \cup E \cup E'} b_i \rhd f_j(\bar{e}) = f_j'(\bar{e}).$$

This follows easily by β-reducing $f_j'(\bar{e})$, because $b_i \wedge \mathrm{tt} \models b_j[\bar{e}/\bar{x}_j]$, by hypothesis.

Cases t_i is $\alpha.t$, $t_1 + t_2$, $t_1|t_2$ and $t\backslash c$ are all similar in that we apply induction to their sub-terms and then use the respective rules for congruence.

Case t_i is $b' \rightarrow t$. We know that if we have $t_i \overset{b}{\gg} (j, \bar{e})$, then we must have $t \overset{b''}{\gg} (j, \bar{e})$ where $b = b' \wedge b''$. This means that we know $b_i \wedge b' \wedge b'' \models b_j[\bar{e}/\bar{x}_j]$ by hypothesis. Induction gives

$$\vdash_{D \cup E \cup E'} b_i \wedge b' \rhd t[\bar{f}/\bar{X}] = t'[\bar{f}'/\bar{X}']$$

and boolean manipulation gives

$$\vdash_{D \cup E \cup E'} b_i \rhd t_i[\bar{f}/\bar{X}] = t_i'[\bar{f}'/\bar{X}'].$$

So, collecting derivations (6.2), (6.3) and the hypothesis together we get

$$\vdash_{D \cup E \cup E'} b_i \rhd f_i'(\bar{x}_i) = t_i'[\bar{f}'/\bar{X}'],$$

whence

$$\vdash_{D \cup E \cup E'} b_1 \rhd f_1'(\bar{x}_1) = X_1'(\bar{x}_1)$$

by UFI-O. The result now follows easily by transitivity and dec-E. ■

6.2 Soundness and completeness for strong bisimulation

Soundness of the proof system of the previous section, with respect to strong bisimulation, is due to Hennessy and Lin. It is clear, because of derivability, that UFI-Inv, is a sound proof technique.

Proposition 6.2.1 *(Soundness) If D is a guarded declaration and $t, u \in \mathcal{T}_D$ then $\mathcal{A} \vdash_D b \rhd t = u$ implies $t \sim_L^b u$.*

Proof See [42] ■

The converse of this is the interesting proposition of completeness. Clearly we cannot expect any general completeness results. However, if we restrict ourselves to ***regular*** processes, that is, recursively defined processes which make no use of parallelism or hiding, then we can obtain completeness results provided that all recursions are guarded. We strengthen Hennessy and Lin's completeness results of [42] by allowing fully parameterised processes; we make no restrictions on the form of parameters.

The exposition of this completeness result uses the symbolic technique but the general approach, however, is not entirely dissimilar from the original completeness proofs of [73].

The first step therefore is to develop some notion of normal form for regular terms. We will find it convenient to work with declarations rather than arbitrary terms. To this end we show that every regular term can be provably transformed into the leading identifier of a regular declaration.

Proposition 6.2.2 *Let t be a regular term with identifiers in regular D. Then there exists a regular declaration $D' = \{X_i \Longleftarrow f_i\}_I$ such that*

$$\vdash_{D \cup D'} \mathbf{tt} \rhd t = X_1(\bar{x}),$$

where $fv(t) = \bar{x}$. Moreover, if D is a guarded declaration then so is D'.

Proof We let D' be the declaration D with the extra definition $X_1 \Longleftarrow \lambda\bar{x}.t^*$ where t^* is the term obtained from t by replacing any unguarded occurrences of identifiers with their definitions from D. It should be clear that D' is guarded if D is, and

$$\vdash_{D \cup D'} \mathbf{tt} \rhd t = X_1(\bar{x})$$

follows easily by β-reduction and FIX. ■

A regular, guarded, declaration $D = \{X_i \Longleftarrow \lambda\bar{x}_i.t_i\}_I$ is said to be a ***standard form*** if each t_i is of the form

$$\sum_{k \in K_i} c_{ik} \rightarrow \sum_{p \in P_{ik}} \alpha_{ikp}.X_{f(ikp)}(\bar{e}_{ikp})$$

such that $\bigvee_k c_{ik} = \mathbf{tt}$ for each i and $c_{ik} \wedge c_{ik'} = \mathbf{ff}$ for $k \neq k'$. We also ask that all input actions in D use the same bound variable.

The proof that every regular declaration may be provably transformed into a standard form is given in detail in [42] and involves using axioms in $\mathcal{A}$ to reorder summations with some simple structural manipulation of booleans. This allows us to now work exclusively, without loss of generality, with standard declarations.

We consider the definition of symbolic bisimulation given in Chapter 2. In particular, we notice that, whenever $t \sim^b u$ holds, each symbolic transition from t may induce a different partition of b. Similarly, every transition from u may induce different partitions of b. Because there are only ever finitely many symbolic transitions from regular terms it is possible to combine the different partitions which are induced into a single, very fine grain, partition. The following lemma from [42] pursues this idea.

Lemma 6.2.3 *Suppose $t \equiv \sum_I \alpha_i.t_i$ and $u \equiv \sum_J \beta_j.u_j$ where $bv(\alpha_i) \cap bv(\beta_j) \cap fv(b,t,u) = \emptyset$. Then $t \sim^b_L u$ if and only if there exists a disjoint b-partition, B, with $fv(B) \subseteq fv(b,t,u)$ such that for each $b' \in B$ we have*

- *For each $i \in I$ there is a $j \in J$ such that $b \models \alpha_i = \beta_j$ and $t_i \sim^{b'}_L u_j$*
- *For each $j \in J$ there is an $i \in I$ such that $b \models \alpha_i = \beta_j$ and $t_i \sim^{b'}_L u_j$.*

Theorem 6.2.4 *Let $D_1 = \{X_i \Longleftarrow f_i\}_I$ and $D_2 = \{Y_j \Longleftarrow g_j\}_J$ be standard declarations such that $X_1(\bar{e}_1) \sim_L^b Y_1(\bar{e}'_1)$. Then there exists a standard declaration $E = \{Z_{ij} \Longleftarrow h_{ij}\}_{I\times J}$ such that*

$$\mathcal{A} \vdash_{D_1 \cup E} b \rhd X_1(\bar{e}_1) = Z_{11}(\bar{e}_1, \bar{e}'_1)$$

and

$$\mathcal{A} \vdash_{D_2 \cup E} b \rhd Y_1(\bar{e}'_1) = Z_{11}(\bar{e}_1, \bar{e}'_1).$$

Proof We assume the declarations to have disjoint parameters and that all input prefixes bind the same variable w, not a parameter of either D_1 or D_2. Thus, suppose

$$X_i \Longleftarrow \lambda\bar{x}_i. \sum_{k\in K_i} c_{ik} \to \sum_{p\in P_{ik}} \alpha_{ikp}.X_{f(ikp)}(\bar{e}_{ikp})$$

and

$$Y_j \Longleftarrow \lambda\bar{y}_j. \sum_{l\in L_j} d_{jl} \to \sum_{q\in Q_{jl}} \beta_{jlq}.X_{g(jlq)}(\bar{e}_{jlq}).$$

We intend to apply the UFI-Inv rule to obtain the two desired sequents. In order to do this we need to find our loop invariants. For the time being we define these abstractly as the collection $\{b_{ij}\}$ of booleans such that

(i) $fv(b_{ij}) \subseteq \bar{x}_i \cup \bar{y}_j$

(ii) $\delta \models b_{ij}$ if and only if $X_i(\bar{x}_i)\delta \sim Y_j(\bar{y}_j)\delta$.

The b_{ij} are the most general booleans which guarantee the given equivalence and, as such, should be suitable as loop invariants. In fact we notice that, in general,

$$X_i(\bar{e}) \sim_L^b Y_j(\bar{e}') \quad \text{implies} \quad b \models b_{ij}[\bar{e}, \bar{e}'/\bar{x}_i, \bar{y}_j]. \tag{6.4}$$

This is easy to see by considering any $\delta \models b$, so, by Theorem 2.5.4, we know that $X_i(\bar{e})\delta \sim_L Y_j(\bar{e})\delta$. Because $\bar{x}_i$ and $\bar{y}_j$ are disjoint, this can be rewritten as

$$X_i(\bar{x}_i)\delta^+ \sim_L Y_j(\bar{y}_j)\delta^+ \tag{6.5}$$

where δ^+ is $\delta[\bar{e}, \bar{e}'/\bar{x}_i, \bar{y}_j]$. But notice that the second defining clause for b_{ij} stipulates that any environment satisfying (6.5) must satisfy b_{ij}. This means that $\delta^+ \models b_{ij}$, whence $\delta \models b_{ij}[\bar{e}, \bar{e}'/\bar{x}_i, \bar{y}_j]$.

The first instance of (6.4) we consider is that $b \models b_{11}[\bar{e}_1, \bar{e}'_1]$. So, to prove the result, it suffices, by using the rule CONS and the derivable rule SUBST, to find the common declaration E such that

$$\vdash_{D_1 \cup E} b_{11} \rhd X_1(\bar{x}_1) = Z_{11}(\bar{x}_1, \bar{y}_1) \tag{6.6}$$

and

$$\vdash_{D_2 \cup E} b_{11} \rhd Y_1(\bar{y}_1) = Z_{11}(\bar{x}_1, \bar{y}_1). \tag{6.7}$$

We let b_{ijkl} be the boolean expression $b_{ij} \wedge c_{ik} \wedge d_{jl}$ for each i, j, k, l and notice that, because of the disjoint guards, that $t_{ik} \sim_L^{b_{ijkl}} u_{jl}$, where

$$t_{ik} \equiv \sum_{p\in P_{ik}} \alpha_{ikp}.X_{f(ikp)}(\bar{e}_{ikp})$$

and

$$u_{jl} \equiv \sum_{q\in Q_{jl}} \beta_{jlq}.Y_{g(jlq)}(\bar{e}_{jlq}).$$

Furthermore, these pairs of terms satisfy the hypothesis of Lemma 6.2.3, so we apply this lemma to obtain the partition B_{ijkl}. Now for each $b' \in B_{ijkl}$ we define

$$I^{b'} = \left\{ (p,q) \mid b' \models \alpha_{ikp} = \beta_{jlq} \text{ and } X_{f(ikp)}(\bar{e}_{ikp}) \sim_L^{b'} Y_{g(jlq)}(\bar{e}_{jlq}) \right\}.$$

The properties of B_{ijkl} given by Lemma 6.2.3 ensure that $I^{b'}$ is a total and surjective relation on $P_{ik} \times Q_{jl}$. We use this relation in the construction of Z_{ij}:

Let E be the standard declaration given by

$$\left\{ Z_{ij} \Longleftarrow \lambda\bar{x}_i\bar{y}_j . \sum_{\substack{k\in K_i \\ l\in L_j}} \sum_{b'\in B_{ijkl}} b' \to V^{b'} \right\}$$

where $V^{b'}$ is $\sum_{(p,q)\in I^{b'}} \alpha_{ikp}.Z_{f(ikp)g(jlq)}(\bar{e}_{ikp},\bar{e}_{jlq})$.

In order to establish the sequent (6.7), we will use UFI-Inv on the declaration E, using the invariants $\{b_{ij}\}$ and the terms $g'_{ij} \equiv \lambda\bar{x}_i\bar{y}_j.Y_j(\bar{y}_j)$. Let us first discharge the side-condition of this rule. We need to show that

$$b_{ij} \wedge b' \models b_{i'j'}[\bar{e}\bar{e}'/\bar{x}_i\bar{y}_j]$$

whenever $(b', i'j', \bar{e}\bar{e}') \in Calls(Z_{ij})$. This follows easily if we consider what $Calls(Z_{ij})$ actually is in this case; for any $(b', i'j', \bar{e}\bar{e}') \in Calls(Z_{ij})$ we must have $X_{i'}(\bar{e}) \sim_L^{b'} Y_{j'}(\bar{e}')$ because each call is determined by the $I^{b'}$ and the construction of this relation guarantees this bisimilarity. Now, by the property (6.4), we know that $b_{ij} \wedge b' \models b' \models b_{i'j'}[\bar{e}\bar{e}'/\bar{x}_i\bar{y}_j]$, as required.

So, it only remains to establish the hypotheses of the UFI-Inv rule, that

$$\vdash_{D_2\cup E} b_{ij} \rhd g'_{ij}(\bar{x}_i,\bar{y}_j) = \sum_{\substack{k\in K_i \\ l\in L_j}} \sum_{b'\in B_{ijkl}} b' \to V^{b'}[\bar{g}'/\bar{Z}].$$

Because $\bigvee_k c_{ik} = \bigvee_l d_{jl} = \mathsf{tt}$, this can be derived from the CASE rule and β-reduction if we can show

$$\vdash_{D_2\cup E} b_{ijkl} \rhd \sum_l d_{jl} \to u_{jl} = \sum_{\substack{k\in K_i \\ l\in L_j}} \sum_{b'\in B_{ijkl}} b' \to V^{b'}[\bar{g}'/\bar{Z}]$$

for each k and l. This can, in turn, be derived from

$$\vdash_{D_2\cup E} b' \rhd u_{jl} = V^{b'}[\bar{g}'/\bar{Z}]$$

for each $b' \in B_{ijkl}$. This relies upon the fact that both the guards d_{jl} and the partition B_{ijkl} are disjoint.

Recall that $V^{b'}$ is $\sum_{(p,q)\in I^{b'}} \alpha_{ikp}.Z_{f(ikp)g(jlq)}(\bar{e}_{ikp},\bar{e}_{jlq})$, so our goal is, in fact,

$$\vdash_{D_2\cup E} b' \rhd \sum_{q\in Q_{jl}} \beta_{jlq}.Y_{g(jlq)}(\bar{e}_{jlq}) = \sum_{(p,q)\in I^{b'}} \alpha_{ikp}.g'_{f(ikp)g(jlq)}(\bar{e}_{ikp},\bar{e}_{jlq}).$$

We immediately see that, by β-reducing g' we can derive this sequent from

$$\vdash_{D_2\cup E} b' \rhd \sum_{q\in Q_{jl}} \beta_{jlq}.Y_{g(jlq)}(\bar{e}_{jlq}) = \sum_{(p,q)\in I^{b'}} \alpha_{ikp}.Y_{g(jlq)}(\bar{e}_{jlq}). \tag{6.8}$$

We then note that $I^{b'}$ is surjective so for each $q \in Q_{jl}$ there is a $p \in P_{ik}$ such that $(p,q) \in I^{b'}$. Thus, for each $q \in Q_{jl}$ there is a p such that $b' \models \alpha_{ikp} = \beta_{jlq}$. It is clear that

$$\vdash_{D_2\cup E} b' \rhd Y_{g(jlq)}(\bar{e}_{jlq}) = Y_{g(jlq)}(\bar{e}_{jlq})$$

so an application of TAU, OUTPUT, or L-INPUT will give us

$$\vdash_{D_2 \cup E} b' \rhd \beta_{jlq}.Y_{g(jlq)}(\bar{e}_{jlq}) = \alpha_{ikp}.Y_{g(jlq)}(\bar{e}_{jlq}).$$

We repeat this for each q and add to obtain (6.8).

The sequent (6.6) is slightly easier to establish using a similar argument owing to the fact that E is *built in favour* of D_1 and each $I^{b'}$ is total. ■

Theorem 6.2.5 *(Completeness) Let t and u be regular terms with identifiers in D, where D is a regular, guarded, declaration. Then*

$$t \sim_L^b u \text{ implies } \mathcal{A} \vdash_D b \rhd t = u.$$

Proof We first transform t and u into declarations by using Proposition 6.2.2. This yields declarations, $D_1 = \{X_i\}$ and $D_2 = \{Y_j\}$ such that

$$\vdash_{D \cup D_1} \mathbf{tt} \rhd t = X_1(\bar{x}) \text{ and } \vdash_{D \cup D_2} \mathbf{tt} \rhd u = Y_1(\bar{y})$$

where $fv(t) = \bar{x}$ and $fv(u) = \bar{y}$. Moreover, we may assume that D_1 and D_2 are standard forms and have disjoint parameters. We know that $X_1(\bar{x}) \sim_L^b Y_1(\bar{y})$ holds by hypothesis and soundness. This allows us to apply Theorem 6.2.4 to obtain an $E = \{Z_{ij}\}$ such that

$$\vdash_{D \cup D_1 \cup E} b \rhd X_1(\bar{x}) = Z_{11}(\bar{x}, \bar{y})$$

and

$$\vdash_{D \cup D_2 \cup E} b \rhd Y_1(\bar{y}) = Z_{11}(\bar{x}, \bar{y}).$$

The rule dec-I and transitivity gives $\vdash_{D \cup D_1 \cup D_2 \cup E} b \rhd t = u$, whence $\vdash_D b \rhd t = u$, by dec-E. ■

6.3 The weak case

Having characterised strong (late) bisimulation over regular processes we consider whether the same techniques apply to characterising weak (late) bisimulation, or more precisely, observation congruence.

We recall the definition of weak bisimulation and observation congruence. In order to state these equivalences we need to make use of the following notation:

- $p \overset{\varepsilon}{\Longrightarrow} p$
- $p \overset{\alpha}{\longrightarrow} q$ implies $p \overset{\alpha}{\Longrightarrow} q$
- $p \overset{\tau}{\longrightarrow} \overset{\alpha}{\Longrightarrow} q$ implies $p \overset{\alpha}{\Longrightarrow} q$
- $p \overset{\tau}{\Longrightarrow} \overset{\tau}{\longrightarrow} q$ implies $p \overset{\tau}{\Longrightarrow} q$
- $p \overset{c!v}{\Longrightarrow} \overset{\tau}{\longrightarrow} q$ implies $p \overset{c!v}{\Longrightarrow} q$

Notice that, because we are using a late semantics, there can be no τ actions after the input transition whenever $p \overset{c?}{\Longrightarrow} (x)t$.

We call a relation $\mathcal{R}$, defined over pairs of value-passing CCS agents, a ***late weak bisimulation*** if for each $(p, q) \in \mathcal{R}$ we have

- whenever $p \overset{c?}{\longrightarrow} (x)t$ then $q \overset{c?}{\Longrightarrow} (y)u$ for some $(y)u$ such that for each $v \in Val$, there is a q' such that $u[v/y] \overset{\varepsilon}{\Longrightarrow} q'$ and $(t[v/x], q') \in \mathcal{R}$.

- whenever $p \xrightarrow{\alpha} p'$ $(\alpha \neq c?)$ then $q \overset{\hat{\alpha}}{\Longrightarrow} q'$ for some q' such that $(p', q') \in \mathcal{R}$

with symmetric conditions for q. We write $p \approx_L q$ if there exists a late weak bisimulation $\mathcal{R}$ such that $(p,q) \in \mathcal{R}$. We will drop the subscript L until we discuss the corresponding *early* equivalence.

Late observation congruence for value-passing CCS, $\cong$, is the relation defined by $p \cong q$ if

- whenever $p \xrightarrow{c?} (x)t$ then $q \overset{c?}{\Longrightarrow} (y)u$ for some $(y)u$ such that for each $v \in Val$, there is a q' such that $u[v/y] \overset{\varepsilon}{\Longrightarrow} q'$ and $t[v/x] \approx q'$.
- whenever $p \xrightarrow{\alpha} p'$ $(\alpha \neq c?)$ then $q \overset{\alpha}{\Longrightarrow} q'$ for some q' such that $p' \approx q'$

along with the same conditions on q.

Extensive use of symbolic semantics for value-passing CCS will be used for the remainder of this chapter. Thus we define late weak symbolic bisimulations and late symbolic congruence for this language.

The symbolic version of the weak transition relation $\Longrightarrow$ is defined as follows:

- $t \overset{\mathbf{tt},\varepsilon}{\Longrightarrow} t$
- $t \overset{b,\alpha}{\longmapsto} u$ implies $t \overset{b,\alpha}{\Longrightarrow} u$
- $t \overset{b,\tau}{\longmapsto} \overset{b',\alpha}{\Longrightarrow} u$ implies $t \overset{b \wedge b',\alpha}{\Longrightarrow} u$
- $t \overset{b,\tau}{\Longrightarrow} \overset{b',\tau}{\longmapsto} u$ implies $t \overset{b \wedge b',\tau}{\Longrightarrow} u$
- $t \overset{b,c!e}{\Longrightarrow} \overset{b',\tau}{\longmapsto} u$ implies $t \overset{b \wedge b',c!e}{\Longrightarrow} u$

Suppose $\mathbf{S} = \{S^b\}$ is a boolean indexed family of relations. Define $\mathcal{WSB}(\mathbf{S})$ to be the family of relations such that

$(t,u) \in \mathcal{WSB}(\mathbf{S})^b$ if whenever $t \overset{b_1,\alpha}{\longmapsto} t'$ there exists a variable z such that $z \notin fv(b,t,u)$ and a $b \wedge b_1$-partition, B, such for each $b' \in B$, $z \notin fv(b')$ and there exists a $u \overset{b_2,\hat{\beta}}{\Longrightarrow} u'$ such that $b' \models b_2$ and

- · if α is τ then $\beta \equiv \tau$ and $(t',u') \in S^{b'}$
- · if α is $c!e$ then $\beta \equiv c!e'$ with $b' \models e = e'$ and $(t',u') \in S^{b'}$
- · if α is $c?x$ then $\beta \equiv c?y$ for some y and there exists a b'-partition B' such that for each $b'' \in B'$ there is a u'' such that $u'[z/y] \overset{b_2',\varepsilon}{\Longrightarrow} u''$ with $b'' \models b_2'$ and $(t'[z/x],u'') \in S^{b''}$.

We call $\{S^b\}$ a ***late weak symbolic bisimulation*** if $S^b \subseteq \mathcal{WSB}(\mathbf{S})^b$ for each b and denote the largest such $\mathbf{S}$ by $\{\approx^b\}$. Once again we now use the definition of $\approx^b$ to define $\cong^b$ the largest congruence contained in $\approx^b$:

$t \cong^b u$ if whenever $t \overset{b_1,\alpha}{\longmapsto} t'$ there exists a variable z such that $z \notin fv(b,t,u)$ and a $b \wedge b_1$-partition, B, such that for each $b' \in B$, $z \notin fv(b')$ and there exists a $u \overset{b_2,\beta}{\Longrightarrow} u'$ such that $b' \models b_2$ and

- · if α is τ then $\beta \equiv \tau$ and $t' \approx^{b'} u'$
- · if α is $c!e$ then $\beta \equiv c!e'$ with $b' \models e = e'$ and $t' \approx^{b'} u'$
- · if α is $c?x$ then $\beta \equiv c?y$ for some y and there exists a b'-partition B' such that for each $b'' \in B'$ there is a u'' such that $u'[z/y] \overset{b_2',\varepsilon}{\Longrightarrow} u''$ with $b'' \models b_2'$ and $t'[z/x] \approx^{b''} u''$

(Symmetric conditions on u have been omitted in the above definitions).

We know from [40] that symbolic congruence captures observation congruence correctly:

Theorem 6.3.1 *For any terms t and u,*

$$t \cong^b u \text{ iff } (\forall \delta \cdot \delta \models b \text{ implies } t\delta \cong u\delta).$$

Proof See [40]. ■

We now consider the proof system of the previous section and ask what modifications are necessary to characterise late observation congruence. Firstly, there is the question of guardedness for guaranteeing the soundness of the UFI rule. Up to now we have simply asked that declarations be guarded, that is, that identifier names in declarations appear within the scope of actions. In the weak setting the internal, τ, actions are abstracted away. For this reason we must ask that identifier names in declarations be guarded by *strong* actions — not τ actions. This is tantamount to asking that there are no τ loops in declarations, which is slightly tricky to formalize. To this end, we say that an identifier is strongly guarded in a term t if every occurrence of X in t occurs in some subexpression $\alpha.t'$ of t where $\alpha \neq \tau$. A term t is called strongly guarded if each identifier in t is strongly guarded in t. Unfortunately strong guardedness is not preserved by τ reduction. To see this we consider the example $p \equiv \tau.(c!v.X|c?x.\mathbf{O})$. Now p is strongly guarded but can τ reduce to $X|\mathbf{O}$ which is not strongly guarded. In [74], Milner avoided this situation by asking that declarations be *sequential* as well as guarded. We say that a term t is ***sequential*** if for each X in t, every subexpression of t which contains X (other than X itself) is of the form $\alpha.t'$ or $t_1 + t_2$. The example process p given above is clearly not sequential because $(c!v.X|c?x.\mathbf{O})$ is a subexpression of p which contains X, but it is not one of the two allowable forms. We will be interested in strongly guarded, sequential declarations.

Given a declaration $D = \{X_i \Longleftarrow \lambda\bar{x}_i.t_i\}_I$, we define the relation $\rightsquigarrow$ on identifiers in D by letting

$$X_i \rightsquigarrow X_j \text{ if } X_j \text{ is not strongly guarded in } t_i$$

and we consider the transitive closure of this relation by defining

$$D \text{ is } \textbf{\textit{strongly guarded}} \text{ if it is guarded and } X_i \not\rightsquigarrow^+ X_i \text{ for each } i \in I.$$

Moreover, D is ***sequential*** if each t_i is a sequential term.

We now assume that the side-condition for the UFI rule demands that the declaration used be *strongly* guarded and sequential.

Clearly we will also need some additional axioms to abstract away the internal actions in the proof system. The well known τ rules from [74] serve this purpose adequately:

T1 : $\alpha.\tau.X = \alpha.X$

T2 : $X + \tau.X = \tau.X$

T3 : $\alpha.(X + \tau.Y) + \alpha.Y = \alpha.(X + \tau.Y)$

We will write $\mathcal{A}_\tau$ to denote the four axioms $\mathcal{A}$ along with the axioms $T1$ through $T3$ above. The notation $\mathcal{A}_\tau \vdash_D b \rhd t = u$ will now mean that the sequent $\vdash_D b \rhd t = u$ can be derived using the proof rules of Figures 6.1, 6.2 and axioms in $\mathcal{A}_\tau$.

The first question we ask about this modified proof system is whether it remains sound for observation congruence. The following results help to prove that this is indeed the case. These are generalisations of Lemma 12 and Proposition 13 of Chapter 7.3 of [74].

Lemma 6.3.2 *Suppose C is a strongly guarded, sequential term with identifiers in $\bar{X}$. Also suppose that $C[\bar{f}/\bar{X}]\delta \xrightarrow{\alpha} p$ for some δ. Then there is a sequential term C' such that $p \equiv C'[\bar{f}/\bar{X}]\delta$ (or $p \equiv (x)C'[\bar{f}/\bar{X}]\delta$ if α is $c?$) and for any $\bar{g}$ we have $C[\bar{g}/\bar{X}]\delta \xrightarrow{\alpha} C'[\bar{g}/\bar{X}]\delta$. Moreover, if α is τ then C' strongly guarded.*

Proof Structural induction on the term C. Sequentiality makes most cases trivial. The case where C is a choice $C_1 + C_2$ follows easily by induction and we consider the case where C is a prefixed term $\beta.C'$. We notice that if α is $c!v$ then $\beta \equiv c!e$ for some e such that $v = [\![e\delta]\!]$. Otherwise α is β. We also notice that $C[\bar{g}/\bar{X}]\delta \xrightarrow{\alpha} C'[\bar{g}/\bar{X}]($ or $(x)C'[\bar{g}/\bar{X}])$. Clearly sequentiality is preserved by reduction and if α is τ then by strong guardedness C' must be strongly guarded also. ■

Proposition 6.3.3 *Suppose $D = \{X_i \Longleftarrow h_i\}_I$ is a strongly guarded, sequential, declaration and suppose $\{f_i, g_i\}_I$ are terms such that $\bar{f} \cong \bar{h}[\bar{f}/\bar{X}]$ and $\bar{g} \cong \bar{h}[\bar{g}/\bar{X}]$. Then $\bar{f} \cong \bar{g}$.*

Proof First we notice that we can *unfold* the expression $\bar{h}[\bar{f}/\bar{X}]$ by replacing each f_j in each h_i with $h_j[\bar{f}/\bar{X}]$. If we repeatedly perform this unfolding we will eventually obtain a sequence of terms $\bar{h}^*$ which are all sequential and strongly guarded, moreover $\bar{f} \cong \bar{h}^*[\bar{f}/\bar{X}]$ by congruence properties of $\cong$. We can unfold $\bar{h}[\bar{g}/\bar{X}]$ in a similar way and can obtain $\bar{h}^*$ such that $\bar{g} \cong \bar{h}^*[\bar{g}/\bar{X}]$ also.

We now define

$$\mathcal{R} = \{C[\bar{f}/\bar{X}]\delta, C[\bar{g}/\bar{X}]\delta \mid \forall \delta, C \in T_D, \text{ sequential}\}$$

and show that $\mathcal{R}$ is a weak bisimulation upto $\approx$, [74], [95]. To do this it is sufficient, by symmetry, to show that $C[\bar{f}/\bar{X}]\delta \overset{\alpha}{\Longrightarrow} p$ implies that $C[\bar{g}/\bar{X}]\delta \overset{\alpha}{\Longrightarrow} q$ for some q such that $p \approx \mathcal{R} \approx q$.

For ease of presentation we will simply write $C[\bar{f}]$ for $C[\bar{f}/\bar{X}]$. Suppose then, that $C[\bar{f}]\delta \overset{\alpha}{\Longrightarrow} p$. We know that $C[\bar{f}] \cong C[\bar{h}^*[\bar{f}]]$, so we have $C[\bar{h}^*[\bar{f}]] \overset{\alpha}{\Longrightarrow} p'$ for some p' such that $p' \approx p$. Note that the term $C[\bar{h}^*]$ is sequential and strongly guarded.

If α is τ then we repeatedly apply Lemma 6.3.2 to obtain a C' such that $p' \equiv C'[\bar{f}]\delta$ and $C[\bar{h}^*[\bar{g}]] \overset{\alpha}{\Longrightarrow} C'[\bar{g}]\delta$. We know that $C[\bar{h}^*[\bar{g}]] \cong C[\bar{g}]$ so we get a transition $C[\bar{g}]\delta \overset{\alpha}{\Longrightarrow} q$ for some q such that $p \approx C'[\bar{f}]\delta\mathcal{R}C'[\bar{g}]\delta \approx q$.

If α is not a τ action we can use Lemma 6.3.2 repeatedly on the transitions

$$C[\bar{h}^*[\bar{f}]\delta \overset{\varepsilon}{\Longrightarrow}\xrightarrow{\alpha} C'[\bar{f}]\delta \overset{\varepsilon}{\Longrightarrow} p'$$

to obtain

$$C[\bar{h}^*[\bar{g}]\delta \overset{\varepsilon}{\Longrightarrow}\xrightarrow{\alpha} C'[\bar{g}]\delta.$$

Now $C'[\bar{f}]\delta\mathcal{R}C'[\bar{g}]\delta$ and $C'[\bar{f}]\delta \overset{\varepsilon}{\Longrightarrow} p'$ so by the previous case where α is τ we find $C'[\bar{g}]\delta \overset{\varepsilon}{\Longrightarrow} q$ such that $p \approx p' \approx \mathcal{R} \approx q$. ■

Theorem 6.3.4 *(Soundness) If D is a declaration and $t, u \in \mathcal{T}_D$ then $\mathcal{A}_\tau \vdash_D b \rhd t = u$ implies $t \cong^b u$*

Proof The proof proceeds by induction on the length of the derivation. The base cases involve checking that each axiom in $\mathcal{A}_\tau$ is sound. Otherwise we consider the last proof rule used in the derivation of $\vdash_D b \rhd t = u$. The only difficult case is where the UFI rule was last used. This case is dealt with by the previous proposition. ■

6.4 Completeness for observation congruence

We aim to prove the converse of Theorem 6.3.4. Once again we restrict our attention to regular terms. It is useful to note that any regular term must immediately be sequential. The approach to proving completeness follows the example of the strong case — we find standard forms for terms and show provability for these. In this case however we manipulate the standard forms a little more before we engage on the completeness proof proper. Given a standard declaration $D = \{X_i \Longleftarrow \lambda\bar{x}_i.t_i\}_I$ where each t_i is of the form

$$\sum_{k \in K_i} c_{ik} \rightarrow \sum_{p \in P_{ik}} \alpha_{ikp}.X_{f(ikp)}(\bar{e}_{ikp}),$$

then we say that D is ***saturated*** if

$$X_i(\bar{e}) \overset{b,\alpha}{\Longrightarrow} X_j(\bar{e}') \text{ implies } X_i(\bar{e}) \overset{b,\alpha}{\longmapsto} X_j(\bar{e}').$$

Proposition 6.4.1 *Every regular declaration D, can be provably transformed into a standard, saturated, declaration.*

Proof We know from [42] that every regular declaration can be made standard. We show here that saturation is possible. It is enough to prove the following lemma. ■

Lemma 6.4.2 *(Absorption) If D is the standard form described above and $X_i(\bar{e}) \overset{b,\alpha}{\Longrightarrow} X_j(\bar{e}')$ with $fv(b) \cap bv(\alpha) = \emptyset$, then*

$$\mathcal{A}_\tau \vdash_D X_i(\bar{e}) = X_i(\bar{e}) + b \rightarrow \alpha.X_j(\bar{e}').$$

Proof We use induction on the length of the transition $X_i(\bar{e}) \overset{b,\alpha}{\Longrightarrow} X_j(\bar{e}')$.

Case $X_i(\bar{e}) \overset{b,\alpha}{\longmapsto} X_j(\bar{e}')$. Then we know that b is $c_{ik}[\bar{e}/\bar{x}_i]$ for some ik and that $\alpha \equiv \alpha_{ikp}[\bar{e}/\bar{x}_i]$ for some p such that $f(ikp) = j$, and that $\bar{e}' \equiv \bar{e}_{ikp}[\bar{e}/\bar{x}_i]$. It should be clear that

$$\vdash X_i(\bar{x}_i) = X_i(\bar{x}_i) + c_{ik} \rightarrow \alpha_{ikp}.X_j(\bar{e}_{ikp})$$

and then by the derived SUBST rule we get our result.

Case $X_i(\bar{e}) \overset{b_1,\tau}{\Longrightarrow} X_{j'}(\bar{e}'') \overset{b_2,\alpha}{\longmapsto} X_j(\bar{e}')$ with $b = b_1 \wedge b_2$. We know by induction that

$$\vdash X_i(\bar{e}) = X_i(\bar{e}) + b_1 \rightarrow \tau.X_{j'}(\bar{e}'')$$

and

$$\vdash X_{j'}(\bar{e}'') = X_{j'}(\bar{e}'') + b_2 \rightarrow \alpha.X_j(\bar{e}').$$

Combining these we obtain

$$\vdash X_i(\bar{e}) = X_i(\bar{e}) + b_1 \rightarrow \tau.(X_{j'}(\bar{e}'') + b_2 \rightarrow \alpha.X_j(\bar{e}')).$$

We can now use axiom $T2$ with suitable manipulation of booleans to obtain

$$\vdash X_i(\bar{e}) = X_i(\bar{e}) + b_1 \rightarrow b_2 \rightarrow \alpha.X_j(\bar{e}')$$

from which the result follows easily.

Case $X_i(\bar{e}) \overset{b_1,\alpha}{\Longrightarrow} X_{j'}(\bar{e}'') \overset{b_2,\tau}{\Longrightarrow} X_j(\bar{e}')$ with $b = b_1 \wedge b_2$. We know by induction that

$$\vdash X_i(\bar{e}) = X_i(\bar{e}) + b_1 \rightarrow \alpha.X_{j'}(\bar{e}'')$$

and

$$\vdash X_{j'}(\bar{e}'') = X_{j'}(\bar{e}'') + b_2 \rightarrow \tau.X_j(\bar{e}').$$

Together, these give

$$\vdash X_i(\bar{e}) = X_i(\bar{e}) + b_1 \rightarrow \alpha.(X_{j'}(\bar{e}'') + b_2 \rightarrow \tau.X_j(\bar{e}')).$$

We know that $fv(b_2) \cap bv(\alpha) = \emptyset$ so we can use a derivable form of axiom $T3$ decorated with booleans, [41], to give us

$$\vdash X_i(\bar{e}) = X_i(\bar{e}) + b_1 \rightarrow \alpha.(X_{j'}(\bar{e}'') + b_2 \rightarrow \tau.X_j(\bar{e}')) + b_1 \rightarrow b_2 \rightarrow \alpha.X_j(\bar{e}'),$$

whence

$$\vdash X_i(\bar{e}) = X_i(\bar{e}) + b_1 \rightarrow b_2 \rightarrow \alpha.X_j(\bar{e}').$$

Once again, the result follows easily. ■

We now present a technical lemma which will assist in the proof of completeness. This is is a version of Lemma 6.2.3 suitable for weak bisimulation equivalence.

Lemma 6.4.3 *Suppose we have standard, saturated declarations*

$$X_i \Longleftarrow \lambda\bar{x}_i.\sum_{k\in K_i} c_{ik} \to \sum_{p\in P_{ik}} \alpha_{ikp}.X_{f(ikp)}(\bar{e}_{ikp})$$

and

$$Y_j \Longleftarrow \lambda\bar{y}_j.\sum_{l\in L_j} d_{jl} \to \sum_{q\in Q_{jl}} \beta_{jlq}.X_{g(jlq)}(\bar{e}_{jlq}).$$

Also suppose that $X_i(\bar{x}_i) \approx^{b\wedge c_{ik}\wedge d_{jl}} Y_j(\bar{y}_j)$*, then* $t_{ik} \approx^{b\wedge c_{ik}\wedge d_{jl}} u_{jl}$ *where*

$$t_{ik} \equiv \sum_{P_{ik}} \alpha_{ikp}.X_{f(ikp)}(\bar{e}_{ikp})$$

and

$$u_{jl} \equiv \sum_{Q_{jl}} \beta_{jlq}.Y_{g(jlq)}(\bar{e}_{jlq}).$$

Moreover there exist disjoint $b\wedge c_{ik}\wedge d_{jl}$*-partitions* $B^{c!}_{ijkl}, B^{c?}_{ijkl}$ *and* B^{τ}_{ijkl} *such that*

- *For each* $b' \in B^{c!}_{ijkl}$ *and for each* $p\in P_{ik}$ *such that* $\alpha_{ikp} \equiv c!e$*, there exists a* $q\in Q_{jl}$ *such that* $\beta_{jlq} \equiv c!e'$ *with* $b' \models e = e'$ *and* $X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b'} Y_{g(jlq)}(\bar{e}_{jlq})$.
- *For each* $b' \in B^{\tau}_{ijkl}$ *and for each* $p\in P_{ik}$ *such that* $\alpha_{ikp} \equiv \tau$*, then either* $X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b'} Y_j(\bar{y}_j)$ *or there exists a* $q\in Q_{jl}$ *such that* $\beta_{jlq} \equiv \tau$ *with* $X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b'} Y_{g(jlq)}(\bar{e}_{jlq})$
- *For each* $b' \in B^{c?}_{ikjl}$ *and for each* $p\in P_{ik}$ *such that* $\alpha_{ikp} \equiv c?w$*, there exists a* $q\in Q_{jl}$ *such that* $\beta_{jlq} \equiv c?w$ *and there exists a disjoint* b'*-partition,* $B'_{p,b'}$ *such that for each* $b'' \in B'_{p,b'}$ *we have* $X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b''} Y_{g(jlq)}(\bar{e}_{jlq})$ *or* $Y_{g(jlq)}(\bar{e}_{jlq}) \overset{d,\tau}{\longmapsto} Y_{j(b'')}(\bar{e}(b''))$ *for some* $j(b'')$ *and* $\bar{e}(b'')$ *with* $b'' \models d$ *and* $X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b''} Y_{j(b'')}(\bar{e}(b''))$

(Similar conditions for each $q \in Q_{jl}$ *follow by symmetry).*

Proof We will write b_{ijkl} as an abbreviation for $b\wedge c_{ik}\wedge d_{jl}$. We know that $X_i(\bar{x}_i) \approx^{b_{ijkl}} Y_j(\bar{y}_j)$ so, by disjointness of the c_{ik}s and d_{jl}s we easily see that $t_{ik} \approx^{b_{ijkl}} u_{jl}$. Choose some $p\in P_{ik}$ and consider the three cases of the form of α_{ikp}.

Case α_{ikp} is $c!e$. We know that $X_i(\bar{x}_i) \approx^{b_{ijkl}} Y_j(\bar{y}_j)$ and that $X_i(\bar{x}_i) \overset{c_{ik},c!e}{\longmapsto} X_{f(ikp)}(\bar{e}_{ikp})$. This means that there exists a b_{ijkl}-partition, $B^{c!}_p$ such that for each $b_p \in B^{c!}_p$ there exists a u' such that $Y_j(\bar{y}_j) \overset{d,c!e'}{\Longrightarrow} u'$ such that $b_p \models e = e'$, $b_p \models d$ and $X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b_p} u'$. But we know, by saturation, that, for some $q\in Q_{jl}$, we must have $d \equiv d_{jlq}$, $\beta_{jlq} \equiv c!e'$, and $u' \equiv Y_{g(jlq)}$.

Case α_{ikp} is τ. Similarly we know that there exists a b_{ijkl}-partition, B^{τ}_p, and for each $b_p \in B^{\tau}_p$ the matching move may either be empty, in which case $X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b_p} Y_j(\bar{y}_j)$, or non-empty, which, by saturation, reduces to a single τ-transition so that $X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b_p} Y_{g(jlq)}(\bar{e}_{jlq})$ for some q.

Case α_{ikp} is $c?w$. By similar reasoning we know that there exists a b_{ijkl}-partition, $B^{c?}_p$ such that for each $b_p \in B^{c?}_p$ we obtain a further partition, B'_{p,b_p} with the relevant matching transitions.

Symmetrically, we can obtain the partitions B^{α}_q. Given these we define $B^{c!}_{ijkl}$ as follows: Let $\{p_1,\dots p_n\}$ denote the set of all $p\in P_{ik}$ such that α_{ikp} is of the form $c!e$. Let

$$D^{c!} = \left\{ \bigwedge_{1\le i\le n} b_i \mid b_i \in B^{c!}_{p_i}, 1\le i\le n \right\}.$$

Similarly, if $\{q_1,\ldots,q_m\}$ is the set of all $q \in Q_{jl}$ such that β_{jlq} is of the form $c!e$ then we let

$$E^{c!} = \left\{ \bigwedge_{1\leq i\leq m} b_i \mid b_i \in B^{c!}_{q_i}, 1 \leq i \leq m \right\}.$$

The partition $B^{c!}_{ijkl}$ will contain conjunctions of booleans chosen pairwise from $D^{c!}$ and $E^{c!}$. Formally, this is

$$B^{c!}_{ijkl} = \left\{ b \wedge b' \mid b \in D^{c!}, b' \in E^{c!} \right\}.$$

It is a simple matter to check that $B^{c!}_{ijkl}$ is indeed a b_{ijkl}-partition. Moreover, given any $b' \in B^{c!}_{ijkl}$ and any $p \in P_{ik}$ such that $\alpha_{ikp} \equiv c!e$, we know that p is represented in b' by one of the p_is. This means that $b' \models b_p$ for some $b_p \in B^{c!}_p$ and the required properties of b' follow easily. We define the partitions B^{τ}_{ijkl} and $B^{c?}_{ijkl}$ in an identical manner. For $b' \in B^{c?}_{ijkl}$ and $p \in P_{ik}$ we know that $b' \models b_p$ for some $b_p \in B^{c?}_p$ and that b_p is further partitioned into B'_{p,b_p}. We actually need a b'-partition called $B'_{p,b'}$ here, but this is obtained simply by defining $\left\{ b'' \wedge b' \mid b'' \in B'_{p,b_p} \right\}$.

It is a trivial exercise to make all of the B^{α}_{ijkl} partitions disjoint. ■

The proof of the theorem we are about to present will be similar to that of the previous section, Theorem 6.2.4. That is, given two declarations, D_1, D_2, we build a third declaration, E, which exhibits common behaviour and use UFI-Inv to show that each of D_1 and D_2 are provably equivalent to E. The terms used in the application of UFI-Inv previously were essentially the identifiers of D_1 (and D_2 respectively). The fact that observation congruence is defined in terms of weak bisimulation complicates matters here. In [73], Milner showed that for substitutivity, the terms needed in the application of UFI sometimes needed to be prefixed by a τ action. He used the terms $\{f_i\}$ where each f_i is either an identifier X of D_1 or is of the form $\tau.X$. We cannot rely on this neat solution because in one boolean world we may need to use X_i as f_i and in another we may need to use $\tau.X_i$. This problem is easily rectified by noticing that, by virtue of axiom $T2$, Milner could have easily used $X + \tau.X$ instead of the term $\tau.X$. We exploit this idea later in the proof of Theorem 6.4.5 but first we show a couple more useful facts.

Lemma 6.4.4 *If $\{b_i\}_I$ is a disjoint b-partition and $fv(b) \cap bv(\alpha) = \emptyset$, then*

(i) $\mathcal{A}_\tau \vdash b \rhd \tau.(t + \sum_{i\in I'} b_i \rightarrow \tau.t) = \tau.t$ *for any $I' \subseteq I$*

(ii) $\vdash b \rhd \sum_{i\in I} b_i \rightarrow \tau.u_i = \tau.\sum_{i\in I} b_i \rightarrow u_i$

Proof For (i), we choose an arbitrary $i \in I$ and show that $\vdash b_i \rhd \tau.(t + \sum_{i\in I'} b_i \rightarrow \tau.t) = \tau.t$. An application of CASE gives the result. Firstly we notice that

$$\vdash b_i \rhd \tau.(t + \sum_{j\in I'} b_j \rightarrow \tau.t) = \tau.(t + \sum_{j\in I'} b_i \wedge b_j \rightarrow \tau.t).$$

Now there are two cases to consider: if $i \notin I'$ then each $b_i \wedge b_j$ is ff, by disjointness. It is easily seen that, in this case, the result holds. Otherwise, $i \in I'$ and $b_i \wedge b_j =$ ff for each $j \neq i$. Thus

$$\vdash b_i \rhd \tau.(t + \sum_{j\in I'} b_j \rightarrow \tau.t) = \tau.(t + b_i \rightarrow \tau.t).$$

Simple boolean manipulation gives

$$\vdash b_i \rhd \tau.(t + \sum_{j\in I'} b_j \rightarrow \tau.t) = \tau.(t + \tau.t),$$

from which, using $T1$ and $T2$ we achieve the result.

Part (ii) follows by showing

$$\vdash b_i \rhd \sum_{i\in I} b_i \to \tau.u_i = \tau.\sum_{i\in I} b_i \to u_i$$

for each $i \in I$ — the result follows by CASE. This is simple to establish by noticing

$$\begin{aligned}\vdash b_i \rhd \textstyle\sum_{j\in I} b_j \to \tau.u_j &= \textstyle\sum_{j\in I} b_i \wedge b_j \to \tau.u_j\\ &= b_i \to \tau.u_i\\ &= \tau.b_i \to u_i\\ &= \tau.\textstyle\sum_{j\in I} b_i \wedge b_j \to u_j\\ &= \tau.\textstyle\sum_{j\in I} b_j \to u_j.\end{aligned}$$

■

Theorem 6.4.5 *Let $D_1 = \{X_i \Longleftarrow g_i\}_I$ and $D_2 = \left\{Y_j \Longleftarrow g'_j\right\}_J$ be standard, saturated, strongly guarded declarations such that X_1 does not appear in any g_i and Y_1 does not appear in any g'_j. If $X_1(\bar{e}_1) \cong^b Y_1(\bar{e}'_1)$ then there exists a standard declaration $E = \{Z_{ij} \Longleftarrow h_{ij}\}_{I\times J}$ such that*

$$\mathcal{A}_\tau \vdash_{D_1\cup E} b \rhd X_1(\bar{e}_1) = Z_{11}(\bar{e}_1, \bar{e}'_1)$$

and

$$\mathcal{A}_\tau \vdash_{D_2\cup E} b \rhd Y_1(\bar{e}'_1) = Z_{11}(\bar{e}_1, \bar{e}'_1).$$

Proof As before we assume that the declarations have disjoint parameters and that all input prefixes bind the same variable w, which is not a parameter of either D_1 or D_2. Suppose

$$X_i \Longleftarrow \lambda\bar{x}_i.\sum_{k\in K_i} c_{ik} \to \sum_{p\in P_{ik}} \alpha_{ikp}.X_{f(ikp)}(\bar{e}_{ikp})$$

and

$$Y_j \Longleftarrow \lambda\bar{y}_j.\sum_{l\in L_j} d_{jl} \to \sum_{q\in Q_{jl}} \beta_{jlq}.X_{g(jlq)}(\bar{e}_{jlq}).$$

Our loop invariants b_{ij} will be defined abstractly by

(i) $fv(b_{ij}) \subseteq \bar{x}_i \cup \bar{y}_j$

(ii) $\delta \models b_{ij}$ if and only if $X_i(\bar{x}_i)\delta \approx Y_j(\bar{y}_j)\delta$ for $i, j \neq 1$ and

(iii) $\delta \models b_{11}$ if and only if $X_1(\bar{x}_1)\delta \cong Y_1(\bar{y}_1)\delta$

We immediately see that $b \models b_{11}[\bar{e}_1, \bar{e}'_1/\bar{x}_1, \bar{y}_1]$ so it is enough to prove

$$\vdash_{D_1\cup E} b_{11} \rhd X_1(\bar{x}_1) = Z_{11}(\bar{x}_1, \bar{y}_1)$$

and

$$\vdash_{D_2\cup E} b_{11} \rhd Y_1(\bar{y}_1) = Z_{11}(\bar{x}_1, \bar{y}_1).$$

For each i, j and each $k \in K_i, l \in L_j$, write b_{ijkl} for the boolean $b_{ij} \wedge c_{ik} \wedge d_{jl}$ so that $X_i(\bar{x}_i) \approx^{b_{ijkl}} Y_j(\bar{y}_j)$. We can now apply Lemma 6.4.3 to obtain the B^{α}_{ijkl} partitions of b_{ijkl} with the concomitant properties. These partitions allow us to define the indexing relations used to define E. For each $b' \in B^{c!}_{ijkl}, B^{\tau}_{ijkl}$ or $B^{c?}_{ijkl}$, we let

$$I^{c!}_{b'} = \left\{(p,q) \mid b' \models \alpha_{ikp} = \beta_{jlq} \text{ and } X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b'} Y_{g(jlq)}(\bar{e}_{jlq})\right\}$$

$$\begin{aligned} I^{\tau}_{b'} &= \left\{(p,\tau) \mid \text{ if } \alpha_{ikp} \equiv \tau \text{ and } X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b'} Y_j(\bar{y}_j)\right\} \\ &\cup \left\{(\tau,q) \mid \text{ if } \beta_{jlq} \equiv \tau \text{ and } X_i(\bar{x}_i) \approx^{b'} Y_{g(jlq)}(\bar{e}_{jlq})\right\} \\ &\cup \left\{(p,q) \mid \alpha_{ikp} = \beta_{jlq} \equiv \tau \text{ and } X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b'} Y_{g(jlq)}(\bar{e}_{jlq})\right\} \end{aligned}$$

$$\begin{aligned} I^{c?}_{b'} &= \left\{(p, B'_{p,b'}) \mid \alpha_{ikp} \equiv c?w\right\} \\ &\cup \left\{(B'_{q,b'}, q) \mid \beta_{jlq} \equiv c?w\right\} \end{aligned}$$

The partitions $B'_{p,b'}$ referred to in the definition of $I^{c?}_{b'}$ are those provided by Lemma 6.4.3 for matching $c?$ moves. We notice that the properties of the B^{α}_{ijkl} partitions guarantee that each $p \in P_{ik}$ and each $q \in Q_{jl}$ are related to something in one of the $I^{\alpha}_{b'}$ relations.

We are now in a position to describe the common declaration E. We let

$$E = \left\{ Z_{ij} \Longleftarrow \lambda\bar{x}_i\bar{y}_j. \sum_{\substack{k \in K_i \\ l \in L_j}} V^{\tau}_{ijkl} + \sum_{c} (V^{c!}_{ijkl} + V^{c?}_{ijkl}) \right\}$$

where

$$V^{c!}_{ijkl} = \sum_{b' \in B^{c!}_{ijkl}} b' \rightarrow \sum_{(p,q) \in I^{c!}_{b'}} \alpha_{ikp}.Z_{f(ikp)g(jlq)}(\bar{e}_{ikp}, \bar{e}_{jlq})$$

$$\begin{aligned} V^{\tau}_{ijkl} = \sum_{b' \in B^{\tau}_{ijkl}} b' \rightarrow & \sum_{(p,q) \in I^{\tau}_{b'}} \tau.Z_{f(ikp)g(jlq)}(\bar{e}_{ikp}, \bar{e}_{jlq}) \\ + & \sum_{(p,\tau) \in I^{\tau}_{b'}} \tau.Z_{f(ikp)j}(\bar{e}_{ikp}, \bar{y}_j) \\ + & \sum_{(\tau,q) \in I^{\tau}_{b'}} \tau.Z_{ig(jlq)}(\bar{x}_i, \bar{e}_{jlq}) \end{aligned}$$

$$\begin{aligned} V^{c?}_{ijkl} = \sum_{b' \in B^{c?}_{ijkl}} b' \rightarrow & \sum_{(p, B'_{p,b'}) \in I^{c?}_{b'}} c?w. \sum_{b'' \in B'_{p,b'}} b'' \rightarrow Z_{f(ikp)j(b'')}(\bar{e}_{ikp}, \bar{e}(b'')) \\ + & \sum_{(B'_{q,b'}, q) \in I^{c?}_{b'}} c?w. \sum_{b'' \in B'_{q,b'}} b'' \rightarrow Z_{i(b'')g(jlq)}(\bar{e}(b''), \bar{e}_{jlq}). \end{aligned}$$

Notice that E is a strongly guarded declaration because both D_1 and D_2 are strongly guarded. This will allow us to conclude

$$\vdash_{D_2 \cup E} b_{11} \rhd X_{11}(\bar{y}_1) = Z_{11}(\bar{x}_1, \bar{y}_1)$$

as a valid application of UFI-Inv if we can establish the hypotheses required. The other sequent required can be derived in a similar manner. In order to apply UFI-Inv we need to identify which terms $\{f_i\}_I$ to use. We have mentioned already that these cannot simply be the identifiers of D_1, but must account for some possible extra τ actions. We define the terms as follows

$$f_{ij} = \lambda\bar{x}_i\bar{y}_j. \left[X_i(\bar{x}_i) + \sum_{\substack{k \in K_i \\ l \in L_j}} \sum_{b' \in B^{\tau}_{ijkl}} \sum_{(\tau,q) \in I^{\tau}_{b'}} b' \rightarrow \tau.X_i(\bar{x}_i). \right]$$

We know that $X_1(\bar{x}_1) \cong^{b_{11}} Y_1(\bar{y}_1)$ so by definition, there are no (τ, q) entries in the $I^{\tau}_{b'}$ relation for any b' in any B^{τ}_{1k1l}, thus $\vdash_{D_1} f_{11}(\bar{x}_1, \bar{y}_1) = X_1(\bar{x}_1)$ by β reduction. In fact, we notice a more general property of the f_{ij} terms:

$$\vdash_{D_1} b_0 \rhd \alpha.f_{ij}(\bar{e}, \bar{e}') = \alpha.X_i(\bar{e}) \quad \text{if } b_0 \models b_{ij}[\bar{e}, \bar{e}'/\bar{x}_i, \bar{y}_j] \tag{6.9}$$

To prove this we first notice that

$$\vdash_{D_1} b_{ijkl} \rhd \alpha.\tau.\left[X_i(\bar{x}_i) + \sum_{b' \in B^{\tau}_{ijkl}} \sum_{(\tau,q) \in I^{\tau}_{b'}} b' \to \tau.X_i(\bar{x}_i) \right] = \alpha.\tau.X_i(\bar{x}_i)$$

follows directly from Lemma 6.4.4, part (i), and the respective prefixing rules. The guards $\{c_{ik}\}_{K_i}$ and $\{d_{jl}\}$ are disjoint so we can harmlessly add in all of the terms given by K_i and L_j to get

$$\vdash_{D_1} b_{ijkl} \rhd \alpha.\tau.\left[X_i(\bar{x}_i) + \sum_{\substack{k \in K_i \\ l \in L_j}} \sum_{b' \in B^{\tau}_{ijkl}} \sum_{(\tau,q) \in I^{\tau}_{b'}} b' \to \tau.X_i(\bar{x}_i) \right] = \alpha.\tau.X_i(\bar{x}_i),$$

but this is just $\vdash_{D_1} b_{ijkl} \rhd \alpha.\tau.f_{ij}(\bar{x}_i, \bar{y}_j) = \alpha.\tau.X_i(\bar{x}_i)$. This is true for each k and l so CASE and $T1$ will give us $\vdash_{D_1} b_{ij} \rhd \alpha.f_{ij}(\bar{x}_i, \bar{y}_j) = \alpha.X_i(\bar{x}_i)$ and (6.9) follows by SUBST and CONS.

Let us now show that the side-condition of UFI-Inv holds, with this definition of f_{ij}. Consider an arbitrary element of $Calls(Z_{ij})$. We know, by construction, that this must have the form $(b', i'j', \bar{e}\bar{e}')$ where $X^{!}_{i}(\bar{e}) \approx^{b'} Y^{!}_{j}(\bar{e})'$. From this we know that $b' \models b_{i'j'}[\bar{e}, \bar{e}'/\bar{x}_{i'}, \bar{y}_{j'}]$ by definition of $b_{i'j'}$ (note that $i'j'$ cannot be 11 here by assumption). This proves the side-condition of UFI-Inv, so all that remains to do is establish the hypotheses of this proof rule. This requires us to show

$$\vdash_{D_1 \cup E} b_{ij} \rhd f_{ij}(\bar{x}_i, \bar{y}_j) = \sum_{\substack{k \in K_i \\ l \in L_j}} V^{\tau}_{ijkl} + \sum_{c} (V^{c!}_{ijkl} + V^{c?}_{ijkl})[\bar{f}/\bar{Z}].$$

If we express

$$\sum_{p \in P_{ik}} \alpha_{ikp}.X_{f(ikp)}(\bar{e}_{ikp})$$

as the sum of terms $t^{\tau} + \sum_c (t^{c!} + t^{c?})$ in the obvious way then, by disjointness of the guards we can reduce our obligation to the goals.

$$\vdash b_{ijkl} \rhd V^{c!}_{ijkl}[\bar{f}/\bar{Z}] = t^{c!} \tag{6.10}$$

$$\vdash b_{ijkl} \rhd V^{c?}_{ijkl}[\bar{f}/\bar{Z}] = t^{c?} \tag{6.11}$$

and

$$\vdash b_{ijkl} \rhd V^{\tau}_{ijkl}[\bar{f}/\bar{Z}] = t^{\tau} + \sum_{\substack{k \in K_i \\ l \in L_j}} \sum_{b' \in B^{\tau}_{ijkl}} \sum_{(\tau,q) \in I^{\tau}_{b'}} b' \to \tau.X_i(\bar{x}_i) \tag{6.12}$$

The easiest of these is (6.10) so we attend to this first.

We take an arbitrary $b' \in B^{c!}_{ijkl}$ and show $\vdash b' \rhd V^{c!}_{ijkl}[\bar{f}/\bar{Z}] = t^{c!}$. Then, we obtain (6.10) as an instance of the rule CASE. So,

$$\begin{aligned}
\vdash b' \rhd V^{c!}_{ijkl}[\bar{f}/\bar{Z}] &= \sum_{(p,q) \in I^{c!}_{b'}} \alpha_{ikp}.Z_{f(ikp)g(jlq)}(\bar{e}_{ikp}, \bar{e}_{jlq})[\bar{f}/\bar{Z}] \\
&= \sum_{(p,q) \in I^{c!}_{b'}} \alpha_{ikp}.f_{f(ikp)g(jlq)}(\bar{e}_{ikp}, \bar{e}_{jlq}) \\
&= \sum_{(p,q) \in I^{c!}_{b'}} \alpha_{ikp}.X_{f(ikp)}(\bar{e}_{ikp}), \text{ by (6.9).} \\
&= t^{c!}.
\end{aligned}$$

The last step holds because each $p \in P_{ik}$ such that α_{ikp} is some $c!e$ appears in $I^{c!}_{b'}$.

Similarly, we now choose an arbitrary $b' \in B^{c?}_{ijkl}$. We should note that $w \notin fv(b')$ because we are considering a late symbolic equivalence. We aim to show that $\vdash b' \rhd V^{c?}_{ijkl}[\bar{f}/\bar{Z}] = t^{c?}$ and use CASE to yield (6.11). Write $V^{c?}_{ijkl}$ as $\sum_{b' \in B^{c?}_{ijkl}} b' \to V_1 + V_2$ where

$$V_1 = \sum_{(p, B'_{p,b'}) \in I^{c?}_{b'}} c?w. \sum_{b'' \in B'_{p,b'}} b'' \to Z_{f(ikp)j(b'')}(\bar{e}_{ikp}, \bar{e}(b''))$$

and

$$V_2 = \sum_{(B'_{q,b'}, q) \in I^{c?}_{b'}} c?w. \sum_{b'' \in B'_{q,b'}} b'' \to Z_{i(b'')g(jlq)}(\bar{e}(b''), \bar{e}_{jlq}).$$

We now see that

$$\begin{aligned} \vdash b' \rhd V_1[\bar{f}/\bar{Z}] &= \sum_{(p, B'_{p,b'}) \in I^{c?}_{b'}} c?w. \sum_{b'' \in B'_{p,b'}} b'' \to f_{f(ikp)j(b'')}(\bar{e}_{ikp}, \bar{e}(b'')) \\ &= \sum_{(p, B'_{p,b'}) \in I^{c?}_{b'}} c?w. \sum_{b'' \in B'_{p,b'}} b'' \to \tau.X_{f(ikp)}(\bar{e}_{ikp}) \end{aligned}$$

(This follows from the axiom $T1$, Lemma 6.4.4 part (ii), (6.9) and the fact that $b'' \models b_{f(ikp)i(b'')}[\bar{e}_{ikp}, \bar{e}(b'')/\bar{x}_{f(ikp)}, \bar{y}_{j(b'')}]$).

$$\begin{aligned} &= \sum_{(p, B'_{p,b'}) \in I^{c?}_{b'}} c?w.b' \to \tau.X_{f(ikp)}(\bar{e}_{ikp}) \\ &= \sum_{(p, B'_{p,b'}) \in I^{c?}_{b'}} c?w.X_{f(ikp)}(\bar{e}_{ikp}) \text{ by } T1 \text{ and } w \notin fv(b') \\ &= t^{c?} \end{aligned}$$

Again, the last step follows from the property that each relevant p occurs in $I^{c?}_{b'}$. We now show that $V_2[\bar{f}/\bar{Z}]$ can be absorbed by $t_{c?}$ in the world b'. Using $T1$, Lemma 6.4.4, part (ii), and (6.9) we know that

$$\vdash b' \rhd V_2[\bar{f}/\bar{Z}] = \sum_{(B'_{q,b'}, q) \in I^{c?}_{b'}} c?w. \sum_{b'' \in B'_{q,b'}} b'' \to \tau.X_{i(b'')}(\bar{e}(b'')) \tag{6.13}$$

Choose any $(B_{q,b'}, q) \in I^{c?}_{b'}$, so we know that, for each $b'' \in B'_{q,b'}$ there is some p such that either $X_{f(ikp)}(\bar{e}_{ikp}) \overset{b'',\tau}{\longmapsto} X_{i(b'')}$, or $i(b'')$ is just $f(ikp)$. For simplicity assume that the former always holds. Thus

$$\vdash X_{f(ikp)}(\bar{e}_{ikp}) = X_{f(ikp)}(\bar{e}_{ikp}) + b'' \to \tau.X_{i(b'')}(\bar{e}(b''))$$

by Lemma 6.4.2. We know that the $B'_{q,b'}$ partition is disjoint so we can add across it and use Lemma 6.4.4, part (ii), to obtain

$$\vdash b'' \rhd X_{f(ikp)}(\bar{e}_{ikp}) = X_{f(ikp)}(\bar{e}_{ikp}) + \tau. \sum_{b'' \in B'_{q,b'}} b'' \to X_{i(b'')}(\bar{e}(b''))$$

for each b'', and so it follows by CASE and L-INPUT that

$$\vdash b' \rhd c?w.X_{f(ikp)}(\bar{e}_{ikp}) = c?w. \left[X_{f(ikp)}(\bar{e}_{ikp}) + \tau. \sum_{b'' \in B'_{q,b'}} b'' \to X_{i(b'')}(\bar{e}(b'')) \right].$$

We can use $T3$ to obtain, for each b',

$$\vdash b' \rhd c?w.X_{f(ikp)}(\bar{e}_{ikp}) = c?w.X_{f(ikp)}(\bar{e}_{ikp}) + c?w. \sum_{b'' \in B'_{q,b'}} b'' \to X_{i(b'')}(\bar{e}(b''))$$

which, by adding in the rest of $t^{c?}$ and using (6.13), gives us

$$\vdash b' \rhd t^{c?} = t^{c?} + V_2[\bar{f}/\bar{Z}].$$

Therefore we have our result because

$$\begin{aligned} \vdash b' \rhd V^{c?}_{ijkl}[\bar{f}/\bar{Z}] &= V_1[\bar{f}/\bar{Z}] + V_2[\bar{f}/\bar{Z}] \\ &= t^{c?} + V_2[\bar{f}/\bar{Z}] \\ &= t^{c?} \end{aligned}$$

Finally, we show (6.12) by demonstrating

$$\vdash b' \rhd V^{\tau}_{ijkl}[\bar{f}/\bar{Z}] = t^{\tau} + \sum_{\substack{k \in K_i \\ l \in L_j}} \sum_{b' \in B^{\tau}_{ijkl}} \sum_{(\tau,q) \in I^{\tau}_{b'}} b' \to \tau.X_i(\bar{x}_i)$$

for each $b' \in B^{\tau}_{ijkl}$. Now,

$$\begin{aligned} \vdash b' \rhd V^{\tau}_{ijkl}[\bar{f}/\bar{Z}] = & \sum_{(p,q) \in I^{\tau}_{b'}} \tau.f_{f(ikp)g(jlq)}(\bar{e}_{ikp}, \bar{e}_{jlq}) \\ & + \sum_{(p,\tau) \in I^{\tau}_{b'}} \tau.f_{f(ikp)j}(\bar{e}_{ikp}, \bar{y}_j) \\ & + \sum_{(\tau,q) \in I^{\tau}_{b'}} \tau.f_{ig(jlq)}(\bar{x}_i, \bar{e}_{jlq}) \\ \\ = & \sum_{(p,q) \in I^{\tau}_{b'}} \tau.X_{f(ikp)}(\bar{e}_{ikp}) \\ & + \sum_{(p,\tau) \in I^{\tau}_{b'}} \tau.X_{f(ikp)}(\bar{e}_{ikp}) \\ & + \sum_{(\tau,q) \in I^{\tau}_{b'}} \tau.X_i(\bar{x}_i) \quad \text{by (6.9).} \\ \\ = & \; t^{\tau} + \sum_{b' \in B^{\tau}_{ijlk}} \sum_{(\tau,q) \in I^{\tau}_{b'}} b' \to \tau.X_i(\bar{x}_i) \text{ by disjointness of } B^{\tau}_{ijkl}. \\ = & \; t^{\tau} + \sum_{\substack{k \in K_i \\ l \in L_j}} \sum_{b' \in B^{\tau}_{ijkl}} \sum_{(\tau,q) \in I^{\tau}_{b'}} b' \to \tau.X_i(\bar{x}_i) \text{ by disjointness of } c_i\text{s and } d_j\text{s.} \end{aligned}$$

■

Theorem 6.4.6 *(Completeness) Let t and u be regular terms with identifiers in D, where D is a regular, strongly guarded, declaration. Then*

$$t \cong^b u \textit{ implies } \mathcal{A}_{\tau} \vdash_D b \rhd t = u.$$

Proof As before we transform t and u into declarations by using Proposition 6.2.2. This yields declarations, $D_1 = \{X_i \Longleftarrow g_i\}$ and $D_2 = \left\{Y_j \Longleftarrow g'_j\right\}$ such that

$$\vdash_{D \cup D_1} \mathrm{tt} \rhd t = X_1(\bar{x}) \text{ and } \vdash_{D \cup D_2} \mathrm{tt} \rhd u = Y_1(\bar{y})$$

where $fv(t) = \bar{x}$ and $fv(u) = \bar{y}$. Moreover, we notice that, by construction, D_1 and D_2 are strongly guarded and that X_1 and Y_1 will never appear in any g_i and g'_j respectively. Also, we can assume that D_1 and D_2 are standard forms and have disjoint parameters. We know that $X_1(\bar{x}) \cong^b Y_1(\bar{y})$ holds by hypothesis. This allows us to apply Theorem 6.4.5 to obtain an $E = \{Z_{ij}\}$ such that

$$\vdash_{D \cup D_1 \cup E} b \rhd X_1(\bar{x}) = Z_{11}(\bar{x}, \bar{y})$$

and

$$\vdash_{D \cup D_2 \cup E} b \rhd Y_1(\bar{y}) = Z_{11}(\bar{x}, \bar{y}).$$

The rule dec-I and transitivity gives $\vdash_{D \cup D_1 \cup D_2 \cup E} b \rhd t = u$, whence $\vdash_D b \rhd t = u$, by dec-E. ■

Recall that, in the proof of Theorems 6.2.4, 6.4.5, we extracted the loop invariants, b_{ij}, by defining these expressions abstractly. A reasonable question one might ask is whether we can define these invariants in a more concrete manner. The answer to that is yes, up to a point. We have already stated that the seemingly appropriate finite models for regular, guarded, terms of our language are symbolic graphs with assignment. Recent studies of these models [65, 78] provide algorithms for calculating the most general boolean expressions which guarantee pairs of nodes of symbolic graphs with assignment to be bisimilar. This is true for strong bisimulation, at least, but modification of these algorithms ought to be able to generate such expressions for weak bisimulation on graphs without τ loops also. The notable feature of the expressions which are generated, however, is that they are not always expressible in first-order predicate logic. Both [65] and [78] require the use of parameterised fixpoints over first-order logic. Therefore the completeness results presented here are relative to judgements about the data-language expressed in first-order predicate logic augmented with greatest fixpoints.

6.5 The early case

In this chapter, we have concentrated on late bisimulation. So we now consider the companion equivalence, early bisimulation. We saw in Chapter 2 that the essential difference between early and late strong symbolic bisimulation is whether we are allowed to partition over bound variables of terms in order to find matching terms. The same holds true for the weak equivalence but we notice that the freedom to partition over bound variables in the early case gives rise to a simpler definition of weak symbolic bisimulation.

Firstly, weak symbolic transitions can be generalised slightly in the early case by defining

$$\begin{array}{rlcl} & t \overset{b,\alpha}{\Longrightarrow} u & \text{implies} & t \overset{b,\alpha}{\Longrightarrow}_e u \\ \text{and} & t \overset{b_1,\alpha}{\Longrightarrow}\overset{b_2,\tau}{\longmapsto} u & \text{implies} & t \overset{b_1 \wedge b_2,\alpha}{\Longrightarrow}_e u \end{array}$$

so that weak input transitions $\overset{b,c?x}{\Longrightarrow}_e$ have possible τ transitions after the input move.

Suppose $\mathbf{S} = \{S^b\}$ is a boolean indexed family of relations. Define $\mathcal{EWSB}(\mathbf{S})$ to be the family of relations such that

$(t,u) \in \mathcal{EWSB}(\mathbf{S})^b$ if whenever $t \overset{b_1,\alpha}{\longmapsto} t'$ there exists a variable z such that $z \notin fv(b,t,u)$ and a $b \wedge b_1$-partition, B, such for each $b' \in B$ there exists a $u \overset{b_2,\beta}{\Longrightarrow}_e u'$ such that $b' \models b_2$, and

- if α is τ then $\beta \equiv \tau$ and $(t',u') \in S^{b'}$
- if α is $c!e$ then $\beta \equiv c!e'$ with $b' \models e = e'$ and $(t',u') \in S^{b'}$
- if α is $c?x$ then $\beta \equiv c?y$ for some y and $(t'[z/x], u'[z/y]) \in S^{b'}$.

We call $\{S^b\}$ an ***early weak symbolic bisimulation*** if $S^b \subseteq \mathcal{EWSB}(\mathbf{S})^b$ for each b and denote the largest such $\mathbf{S}$ by $\{\approx^b_E\}$. We also use the definition of $\approx^b_E$ to define $\cong^b_E$, the largest congruence contained in $\approx^b_E$:

$t \cong^b_E u$ if whenever $t \overset{b_1,\alpha}{\longmapsto} t'$ there exists a variable z such that $z \notin fv(b,t,u)$ and a $b \wedge b_1$-partition, B, such that for each $b' \in B$ there exists a $u \overset{b_2,\beta}{\Longrightarrow}_e u'$ such that $b' \models b_2$ and

· if α is τ then $\beta \equiv \tau$ and $t' \approx^{b'} u'$

· if α is $c!e$ then $\beta \equiv c!e'$ with $b' \models e = e'$ and $t' \approx^{b'} u'$

· if α is $c?x$ then $\beta \equiv c?y$ for some y and $t'[z/x] \approx^{b'} u'[z/y]$.

(There are of course symmetric conditions on u required in the above definitions.)

Unsurprisingly the only changes to the proof system which are required are concerned with input prefixes. The easiest way to modify the proof system to reflect an early semantics is to adapt the rule $L-INPUT$ to be:

$$\frac{\vdash b \rhd \sum_I \tau.t_i = \sum_J \tau.u_j}{\vdash b \rhd \sum_I c?w.t_i = \sum_J c?w.u_j} \quad \text{if } w \notin fv(b),$$

which we will call $E-INPUT$. A consequence of this rule is the axiom

$$Early: \ c?x.t + c?x.u = c?x.t + c?x.u + c?x.(b \to t + \neg b \to u).$$

which is given in [42], and is based on ideas of [81] for the π-calculus. It is easy to see that this axiom fails to be sound with respect to late bisimulation whenever $x \in fv(b)$. This is because, in order to match the symbolic transition

$$c?x.(b \to t + \neg b \to u) \overset{\mathbf{tt},c?x}{\longmapsto} (b \to t + \neg b \to u),$$

the process $c?x.t + c?x.u$ necessarily must partition $\mathbf{tt}$ into $\{b, \neg b\}$. This partition contains the bound variable x, making this an early symbolic bisimulation. So we will write $\mathcal{A}^E \vdash b \rhd t = u$ to mean that $b \rhd t = u$ can be derived using the axioms from $\mathcal{A}$ and the inference rules from Figure 6.1 along with $E-INPUT$. Accordingly, we write $\mathcal{A}^E_\tau \vdash b \rhd t = u$ if we can derive this sequent from $\mathcal{A}_\tau$ with possible uses of $E-INPUT$.

Theorem 6.5.1 *Suppose D is a regular, guarded, declaration and let t and u be regular terms with identifiers in D. Then*

$$\mathcal{A}^E \vdash b \rhd t = u \textit{ iff } t \sim^b_E u.$$

Moreover, if D is strongly guarded then

$$\mathcal{A}^E_\tau \vdash b \rhd t = u \textit{ iff } t \cong^b_E u.$$

Proof Soundness, in both cases is easy to establish by copying the proofs for the late case and checking the extra axiom *Early*. Adapting the completeness proofs for the late bisimulation to work for the early case is more involved so we demonstrate how this is achieved to show completeness with respect to $\cong^b_E$. The strong case can be achieved in a similar manner.

The first change we must make is in Lemma 6.4.3. We now let the third clause, on $B^{c?}$, read

- For each $b' \in B^{c?}_{ikjl}$ and for each $p \in P_{ik}$ such that $\alpha_{ikp} \equiv c?w$, there exists a $q \in Q_{jl}$ such that $\beta_{jlq} \equiv c?w$ and we have $X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b'}_E Y_{g(jlq)}(\bar{e}_{jlq})$ or $Y_{g(jlq)}(\bar{e}_{jlq}) \overset{d,\tau}{\longmapsto} Y_{j_{b'}}(\bar{e}_{b'})$ for some $j_{b'}$ and $\bar{e}_{b'}$ with $b' \models d$ and $X_{f(ikp)}(\bar{e}_{ikp}) \approx^{b'}_E Y_{j_{b'}}(\bar{e}_{b'})$

The proof with this change goes through easily by appealing to saturation. In the proof of Theorem 6.4.5, the definition of the $I^{c?}_{b'}$ relation must be altered to be

$$I^{c?}_{b'} = \{(p,(j_{b'},\bar{e}_{b'})) \mid \alpha_{ikp} \equiv c?w\} \cup \{((i_{b'},\bar{e}_{b'}),q) \mid \beta_{jlq} \equiv c?w\}$$

where $(i_{b'}, \bar{e}_{b'})$ and $(j_{b'}, \bar{e}_{b'})$ denote uniquely chosen relevant matches given by the early version of Lemma 6.4.3 described above. The term $V^{c?}_{ijkl}$ has to be modified of course, this is simply seen as $c_{ik} \wedge d_{jl} \rightarrow V^{c?}_1 + V^{c?}_2$ where

$$V^{c?}_1 = \sum_{\alpha_{ikp} \equiv c?w} c?w. \sum_{b' \in B^{b?}_{ijkl}} b' \rightarrow Z_{f(ikp) j_{b'}}(\bar{e}_{ikp}, \bar{e}_{b'})$$

and

$$V^{c?}_2 = \sum_{\beta_{jlq} \equiv c?w} c?w. \sum_{b' \in B^{b?}_{ijkl}} b' \rightarrow Z_{i_{b'} g(jlq)}(\bar{e}_{b'}, \bar{e}_{jlq}).$$

The proof continues as in Theorem 6.4.5 until we are required to establish the goal (6.11). This can be done in a similar way but with the following differences:

$$\begin{aligned}
\vdash b_{ijkl} \rhd V_1[\bar{f}/\bar{Z}] &= \sum_{\alpha_{ikp} \equiv c?w} c?w. \sum_{b' \in B^{b?}_{ijkl}} b' \rightarrow f_{f(ikp) j_{b'}}(\bar{e}_{ikp}, \bar{e}_{b'}) \\
&= \sum_{\alpha_{ikp} \equiv c?w} c?w. \sum_{b' \in B^{b?}_{ijkl}} b' \rightarrow \tau.X_{f(ikp)}(\bar{e}_{ikp}) \\
&= \sum_{\alpha_{ikp} \equiv c?w} c?w.b_{ijkl} \rightarrow \tau.X_{f(ikp)}(\bar{e}_{ikp}) \\
&= \sum_{\alpha_{ikp} \equiv c?w} c?w.X_{f(ikp)}(\bar{e}_{ikp}) \text{ by } T1 \text{ and } w \notin fv(b_{ijkl}) \\
&= t^{c?}
\end{aligned}$$

Then we show that $V_2[\bar{f}/\bar{Z}]$ can be absorbed by $t_{c?}$ in the world b_{ijkl}. Using $T1$, Lemma 6.4.4, part (ii), and (6.9) we know that

$$\vdash b_{ijkl} \rhd V_2[\bar{f}/\bar{Z}] \quad = \quad \sum_{\beta_{jlq} \equiv c?w} c?w. \sum_{b' \in B^{b?}_{ijkl}} b' \rightarrow X_{i_{b'}}(\bar{e}_{b'}) \tag{6.14}$$

We know that, for each $q \in Q_{jl}$ such that $\beta_{jlq} \equiv c?w$ and for each b' there is some p, dependent upon b', such that either $X_{f(ikp)}(\bar{e}_{ikp}) \overset{d,\tau}{\longmapsto} X_{i_{b'}}$ with $b' \models d$, or $i_{b'}$ is just $f(ikp)$. For simplicity assume that the former always holds. Thus

$$\vdash X_{f(ikp)}(\bar{e}_{ikp}) = X_{f(ikp)}(\bar{e}_{ikp}) + b' \rightarrow \tau.X_{i_{b'}}(\bar{e}_{b'})$$

by Lemma 6.4.2.

We can prefix by τ actions, use $T3$ and add across the partition $B^{c?}_{ijkl}$ to get

$$\vdash \sum_{\alpha_{ikp} \equiv c?w} \tau.X_{f(ikp)}(\bar{e}_{ikp}) = \sum_{\alpha_{ikp} \equiv c?w} \tau.X_{f(ikp)}(\bar{e}_{ikp}) + \sum_{b' \in B^{c?}_{ijkl}} b' \rightarrow \tau.X_{i_{b'}}(\bar{e}_{b'}).$$

Now we know that, after applying Lemma 6.4.4, part (ii),

$$\vdash b_{ijkl} \rhd t^{c?} = t^{c?} + c?w \sum_{b' \in B^{c?}_{ijkl}} b' \rightarrow X_{i_{b'}}(\bar{e}_{b'})$$

is derivable from $E - INPUT$. Notice that this can be done for each q such that $\beta_{jlq} \equiv c?w$ so we add all such sequents together and use (6.14) to obtain our goal,

$$\vdash b_{ijkl} \rhd t^{c?} = t^{c?} + V_2[\bar{f}/\bar{Z}].$$

We now have our result because

$$\begin{aligned}
\vdash b_{ijkl} \rhd V^{c?}_{ijkl}[\bar{f}/\bar{Z}] &= V_1[\bar{f}/\bar{Z}] + V_2[\bar{f}/\bar{Z}] \\
&= t^{c?} + V_2[\bar{f}/\bar{Z}] \\
&= t^{c?}
\end{aligned}$$

No further modifications of the proofs for the late case are necessary and this completes our proof. ■

We have the axioms for restriction:

$$
\begin{array}{lcll}
\mathbf{O}\backslash c & = & \mathbf{O} & \\
(X+Y)\backslash c & = & X\backslash c + Y\backslash c & \\
(b \rightarrow \alpha.X)\backslash c & = & \left\{ \begin{array}{ll} \mathbf{O} & \text{if } \alpha \text{ is } c!e \text{ or } c?x \\ b \rightarrow \alpha.(X\backslash c) & \text{otherwise.} \end{array} \right. &
\end{array}
$$

and, if we let X, Y denote $\sum_I c_i \rightarrow \alpha_i.X_i$ and $\sum_J d_j \rightarrow \beta_j.Y_j$ with $fv(X) \cap bv(\beta_j) = fv(Y) \cap bv(\alpha_i) = \emptyset$ for each i, j, then we have the axiom for parallel composition:

$$
\begin{array}{rcl}
X|Y & = & \sum_I c_i \rightarrow \alpha_i.(X_i|Y) + \sum_J d_j \rightarrow \beta_j.(X|Y_j) \\
& + & \sum_{I\times J} c_i \wedge d_j \rightarrow \{\tau.(X_i|Y_j[e/x]) \mid \alpha_i \equiv c!e, \beta_j \equiv c?x\} \\
& + & \sum_{I\times J} c_i \wedge d_j \rightarrow \{\tau.(X_i[e/x]|Y_j) \mid \alpha_i \equiv c?x, \beta_j \equiv c!e\}\,.
\end{array}
$$

In addition to these we must also reflect the fact that $\cong$ is preserved by these operators. This is done in the two inference rules:

$$
\frac{\vdash_D b \rhd t = u}{\vdash_D b \rhd t\backslash c = u\backslash c} \text{ and } \frac{\vdash_D b \rhd t_1 = u_1 \quad \vdash_D b \rhd t_2 = u_2}{\vdash_D b \rhd t_1|t_2 = u_1|u_2}
$$

Figure 6.3. Expansion laws for parallel and restriction

6.6 Networks of regular processes

The completeness results that we have presented so far in this chapter have all been restricted to the use of regular terms, which we defined as expressions written without parallel composition and restriction. We can improve our completeness results, for strong bisimulation, in this chapter by showing that a network of regular terms, that is, a term of the grammar

$$N := N|N \mid N\backslash c \mid t$$

where t is a regular term, can be *expanded*, upto strong bisimulation, to a regular term itself. Thus a proof that $N_1 \sim N_2$ can be approached by expanding each of N_1 and N_2 and performing the proof on their expansions. This is a standard approach to dealing with parallel composition and restriction in process algebra [74, 51, 6] and suitable, sound, expansion laws have been presented for value-passing CCS in [41, 42]. We give these in Figure 6.3. We write $\mathcal{A}^{Exp}$ for the set of axioms $\mathcal{A}$ along with the extra axiom schemes used for expansion, and we write $\mathcal{A}^{Exp} \vdash_D^N b \rhd t = u$ to mean that $\vdash_D b \rhd t = u$ can be derived using axioms from $\mathcal{A}^{Exp}$ and the extra derivation rules for congruence with respect to $|$ and $\backslash$.

We saw in Section 2.6 that one of the problems concerning the expansion of parallel compositions in [42] is that the class of restricted parameter processes which Hennessy and Lin consider is not closed under expansion. That is, there are regular, restricted parameter processes p and q such that the expansion of $p|q$ cannot be described by a single regular, restricted parameter, process. In our setting we do not restrict the parameters of recursively defined processes and this extra expressive power is used to prove that repeated expansions of regular terms do eventually yield a regular term. Provided that the regular terms produced are still guarded then such a result clearly implies that our completeness results, for strong bisimulation at least, are strengthened to incorporate networks of regular terms.

Unfortunately the result of our expansion, below, of two *strongly* guarded declarations, is not necessarily strongly guarded. The upshot of this is that, until we can show how to transform guarded declarations into strongly guarded declarations by means of eradicating τ-loops, then we cannot use expansion to obtain completeness with respect to observation congruence over networks of regular terms.

Given a declaration $D = \left\{ X_i \Longleftarrow \lambda\bar{x}_i.\ \sum_{k\in K_i} \alpha_{ik}.X_{f(ik)}(\bar{e}_{ik}) \right\}_I$ then we can define the *regular* declaration $D\backslash c$ as

$$\left\{ Z_i \Longleftarrow \lambda\bar{x}_i.\ \sum_{\alpha_{ik}\neq c!,c?} \alpha_{ik}.Z_{f(ik)}(\bar{e}_{ik}) \right\}_I .$$

If, in addition we have the declaration $E = \left\{ Y_j \Longleftarrow \lambda\bar{y}_j.\ \Sigma_{L_j}\ \beta_{jl}.Y_{g(jl)}(\bar{e}_{jl}) \right\}_J$ such that each $\bar{x}_i$ is distinct from each $\bar{y}_j$, then we can define $D|E$ as the *regular* declaration

$$\left\{ \begin{array}{lll} R_{ij} \Longleftarrow & \lambda\bar{x}_i\bar{y}_j. & \Sigma_{K_i}\,\alpha_{ik}.R_{f(ik)j}(\bar{e}_{ik},\bar{y}_j) + \Sigma_{L_j}\,\beta_{jl}.R_{ig(jl)}(\bar{x}_i,\bar{e}_{jl}) \\ & + & \Sigma_{k,l}\left\{\tau.R_{f(ik)g(jl)}(\bar{e}_{ik},\bar{e}_{jl}[e/x]) \mid \alpha_{ik}\equiv c!e, \beta_{jl}\equiv c?x\right\} \\ & + & \Sigma_{k,l}\left\{\tau.R_{f(ik)g(jl)}(\bar{e}_{ik}[e/x],\bar{e}_{jl}) \mid \alpha_{ik}\equiv c?x, \beta_{jl}\equiv c!e\right\} \end{array} \right\}$$

Proposition 6.6.1 *Suppose $D\backslash c$ and $D|E$ are as above, then*

$$\mathcal{A}^{Exp} \vdash^N_{D\cup D\backslash c} X_1(\bar{x}_1)\backslash c = Z_1(\bar{x}_1))$$

and

$$\mathcal{A}^{Exp} \vdash^N_{D\cup E\cup D|E} (X_1(\bar{x}_1)|Y_1(\bar{y}_1)) = R_{11}(\bar{x}_1,\bar{y}_1)$$

Proof We obtain the first sequent by UFI-Inv using the new declaration $D\backslash c$ and the set of terms $\{\lambda\bar{x}_i.X_i.(\bar{x}_i)\backslash c\}_I$. This requires us to show the hypothesis

$$\vdash X_i(\bar{x}_i)\backslash c = \sum_{\alpha_{ik}\neq c!,c?} \alpha_{ik}.X_{f(ik)}(\bar{e}_{ik})\backslash c.$$

This can be done by β-reducing the left-hand side, under the $\backslash$ operator and then using the axioms for $\backslash c$. The latter sequent is proved similarly, we use UFI-Inv with the declaration $D|E$ and the terms $\{\lambda\bar{x}_i\bar{y}_j.X_i(\bar{x}_i)|Y_j(\bar{y}_j)\}_{I\times J}$. The parallel expansion rule and congruence of $|$ provide the result. ■

Corollary 6.6.2 *(Completeness) Suppose D is a regular guarded declaration and let N_1 and N_2 be regular networks with identifiers in D. Then*

$$\mathcal{A}^{Exp} \vdash^N_D b \triangleright N_1 = N_2 \textit{ iff } N_1 \sim^b N_2.$$

Proof Follows from the previous proposition and Theorems 6.5.1 and 6.4.6. ■

6.6.1 Parameterised vs. Unparameterised declarations

An interesting feature of the expansion law for the parallel operator is that every network of regular declarations can be written as an expansion into a single regular declaration. In particular, this is true of declarations, D, which have no parameter variables, that is *unparameterised* declarations. Notice however, that the resulting expansion may be parameterised! The possible parameterisation creeps in at the stage of converting D into a form suitable for applying the expansion law. This reflects the fact that the class of finite symbolic graphs is not closed under parallel composition.

We now seek some kind of inverse to expansion — to show that every regular, parametrised, declaration can be transformed, upto equivalence, into a network of unparameterised declarations. The idea we follow is loosely based on an example due to Stevens, [97]. A *Cell* process is constructed which can receive a piece of data on an internal channel and remember it until some other process asks for it back. Having returned the data the *Cell* resumes its original state, dataless. The *Main* process in the transformation needs to emulate the behaviour of the declaration, however it cannot assume any knowledge of data, or parameters, initially. *Main* must request its parameters from *Cell* before performing any actions of the declaration. Having executed the action from the declaration, *Main* then passes any data that the declaration would store as a parameter over to *Cell* until *Main* resets itself.

Clearly, from this description we cannot expect the transformation to be equivalent to the original declaration up to strong bisimulation. We show that the implementation is correct up to weak bisimulation at least.

Proposition 6.6.3 *Suppose* $D = \{X_i \Longleftarrow \lambda\bar{x}_i.\sum_{k\in K_i} b_{ik} \rightarrow \alpha_{ik}.X_{f(ik)}(\bar{e}_{ik})\}_I$. *There exists an unparameterised regular declaration* D' *and a regular network* $N \in \mathcal{T}_{D'}$ *such that* $X_1(\bar{e}_0) \approx N$.

Proof We assume, for simplicity, that each of the vectors $\bar{x}_i$ are of unit length. The proof can be easily be generalised to arbitrary length vectors. Suppose that $\mathcal{A} = \{c_i, d_i\}_I$ are fresh channel names not appearing in D. Let D' contain the three definitions

$$\begin{array}{lcl} St & \Longleftarrow & c_1!e_0 \\ Main & \Longleftarrow & \sum_{i\in I} c_i?x_i.\ [\sum_{k\in K_i} b_{ik} \rightarrow \alpha_{ik}.d_{f(ik)}!e_{ik}.Main] \\ Cell & \Longleftarrow & \sum_{i\in I}\sum_{k\in K_i} d_{f(ik)}?x_{f(ik)}.c_{f(ik)}!x_{f(ik)}.Cell. \end{array}$$

Then the network N which we require is $(St|Main|Cell)\backslash\mathcal{A}$.

Let C_i denote the term

$$((\sum_{K_i} b_{ik} \rightarrow \alpha_{ik}.d!e_{ik}.Main)|Cell)\backslash\mathcal{A},$$

let C_i' be $(d_i!x_i.M|Cell)\backslash\mathcal{A}$ and let C_i'' be $(M|c_i!x_i.Cell)\backslash\mathcal{A}$. For simplicity we also assume that $\bar{e}_0$ is a closed data expression so we can define our witnessing bisimulation as

$$\begin{array}{rcl} \mathcal{R} & = & \{(X_1(e_0), (St|Main|Cell)\backslash\mathcal{A})\} \\ & \cup & \{(X_i(e), C_i[e/x_i])\} \\ & \cup & \{(X_i(e), C_i'[e/x_i])\} \\ & \cup & \{(X_i(e), C_i''[e/x_i])\}\,. \end{array}$$

We must show that $\mathcal{R}$ is indeed a bisimulation. Suppose $(p,q) \in \mathcal{R}$, there are four cases to consider:

(i) $p \equiv X_1(e_0)$ and $q \equiv (St|Main|Cell)\backslash\mathcal{A}$. If $p \xrightarrow{\alpha} p'$ for some p' then it must be the case that $[\![b_{1k}[e_0/x_1]]\!] = \mathsf{tt}$ for some $k \in K_1$ with $\alpha = \alpha_{1k}[e_0/x_1]$ and $p' \equiv X_{f(1k)}(e_{1k}[e_0/x_1])$. We can match this transition with a τ move

$$q \xrightarrow{\tau} C_1[e_0/x_1]$$

and then notice that $C_1[e_0/x_1] \xrightarrow{\alpha} q'$ where $q' \equiv C'_{f(1k)}[e_{f(1k)}[e_0/x_1]/x_{f(1k)}]$. It is clear that $(p',q') \in \mathcal{R}$.

For the symmetric matching condition, we know that $q\backslash\mathcal{A} \xrightarrow{\tau} C_1[e_0/x_1]$ is the only transition possible from q. We can match this with the empty transition from p, noticing that $(p, C_1[e_0/x_1]) \in \mathcal{R}$.

(ii) $p \equiv X_i(e)$ and $q \equiv C_i[e/x_i]$. Suppose that $p \xrightarrow{\alpha} p'$ for some p'. Again it must be that $[\![b_{ik}[e/x_i]]\!] = \mathsf{tt}$ for some $k \in K_i$ with $\alpha = \alpha_{ik}[e/x_i]$ and $p' \equiv X_{f(ik)}(e_{ik}[e/x_i])$. This means we have an immediate matching transition

$$q \xrightarrow{\alpha} q' \equiv C'_{f(ik)}[e_{f(ik)}[e/x_i]/x_{f(ik)}]$$

with $(p',q') \in \mathcal{R}$.

Similarly, any move from q must have the corresponding move from p available leaving us in states $(X_{f(ik)}(e_{ik}[e/x_i]), C'_{f(ik)}[e_{f(ik)}[e/x_i]/x_{f(ik)}]) \in \mathcal{R}$.

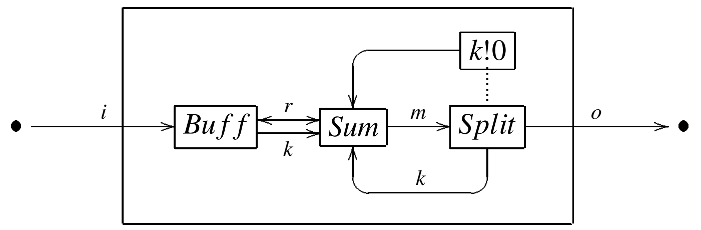


Figure 6.4. Implementation of *Spec*

(iii) $p \equiv X_i(e)$ and $q \equiv C_i'[e/x_i]$. Suppose that $p \xrightarrow{\alpha} p'$ for some p' so that $[[b_{ik}[e/x_i]]] = \mathrm{tt}$ for some $k \in K_i$ with $\alpha = \alpha_{ik}[e/x_i]$ and $p' \equiv X_{f(ik)}(e_{ik}[e/x_i])$. We know that

$$q \xrightarrow{\tau} C_i''[e/x_i] \xrightarrow{\tau} C_i[e/x_i] \xrightarrow{\alpha} q'$$

where $q' \equiv C'_{f(ik)}[e_{f(ik)}[e/x_i]/x_{f(ik)}]$. Thus $(p', q') \in \mathcal{R}$.

The single transition, $q \xrightarrow{\tau} C_i''[e/x_i]$ is matched by an empty transition from p.

(iv) $p \equiv X_i(e)$ and $q \equiv C_i''[e/x_i]$. Suppose that $p \xrightarrow{\alpha} p'$ for some p' so that $[[b_{ik}[e/x_i]]] = \mathrm{tt}$ for some $k \in K_i$ with $\alpha = \alpha_{ik}[e/x_i]$ and $p' \equiv X_{f(ik)}(e_{ik}[e/x_i])$. This time we have

$$q \xrightarrow{\tau} C_i[e/x_i] \xrightarrow{\alpha} q'$$

where $q' \equiv C'_{f(ik)}[e_{f(ik)}[e/x_i]/x_{f(ik)}]$ with $(p', q') \in \mathcal{R}$.

Again, $q \xrightarrow{\tau} C_i[e/x_i]$ is matched by an empty transition from p.

■

6.7 Example

We conclude this chapter with a short example to demonstrate the use of UFI-Inv. Consider the process: $Spec = i?w.\Sigma(0,w)$ where

$$\begin{aligned} \Sigma \quad &\Leftarrow \quad \lambda yy'.[i?w.o!(y+y').\Sigma(y+y',w) \\ &\qquad +o!(y+y').i?w.\Sigma(y+y',w)]. \end{aligned}$$

It is clear that *Spec* specifies a process which receives an input stream on i, and outputs the running total on o. We use two parameters: the first represents the running total and the second, the most recently received input. We implement this process using a parallel composition of three components: a *Sum* process which makes a request for data $*$ on channel r, and subsequently receives two pieces of data, one being the next number on the input stream and the other the last total. Having received these data, in either order, *Sum* passes the sum of them to the process *Split*. The process *Split* receives its input from *Sum* on channel m and simply reroutes this input out on o and back to *Sum*. The input interface is a buffer which can receive on i and pass to k as soon as it has a request to do so from *Sum*. We picture the process in Figure 6.4. The whole process is described syntactically as

$$\begin{aligned} P \quad &= \quad (Sum \mid k!0.Split \mid Buff)\backslash\{k,r,m\} \\ Sum \quad &\Leftarrow \quad r!*.k?x.k?y.m!(x+y).Sum \\ Split \quad &\Leftarrow \quad m?x.o!x.k!x.Split \\ Buff \quad &\Leftarrow \quad r?*.i?x.k!x.Buff. \end{aligned}$$

The diligent reader is invited to check that, using the expansion law for parallel and hiding that P is provably congruent to P_0 where

$$\begin{array}{lcl}
P_0 & \Longleftarrow & \tau.P_1(0) \\
P_1 & \Longleftarrow & \lambda x.[\tau.i?w.\tau\tau.P_2(x+w) + i?w.P_4(x,w)] \\
P_2 & \Longleftarrow & \lambda x.[o!x.\tau.P_1(x) + \tau.[o!x.P_1(x) + i?w.P_3(x,w)]] \\
P_3 & \Longleftarrow & \lambda xx'.[o!x.P_4(x,x') + \tau.o!x.\tau\tau.P_2(x'+x)] \\
P_4 & \Longleftarrow & \lambda xx'.[\tau\tau\tau.P_2(x+x') + \tau\tau\tau.P_2(x'+x)]
\end{array}$$

We can provably simplify this declaration to something more readable. It follows from λ-E and some structural reasoning using the fact $x+x' = x'+x$, that $P_2(x+x') = P_2(x'+x)$ is provable. This allows us, by using axioms $T1$ and $T2$, to prove $\tau.P_1(x) = \tau.i?w.P_2(x+w)$. By similar reasoning we obtain $i?w.P_3(x,x') = i?w.o!x.P_2(x+x')$. Given this then, P can be rendered in saturated, standard form as X_0 where $D_1 =$

$$\left\{\begin{array}{lcl}
X_0 & \Longleftarrow & \tau.X_1(0) + i?w.X_2(0+w) \\
X_1 & \Longleftarrow & \lambda x.i?w.X_2(x+w) \\
X_2 & \Longleftarrow & \lambda x.[o!x.X_1(x) + \tau.X_3(x) + i?w.X_4(x,w)] \\
X_3 & \Longleftarrow & \lambda x.[o!x.X_1(x) + i?w.X_4(x,w)] \\
X_4 & \Longleftarrow & \lambda xx'.o!x.X_2(x+x')
\end{array}\right\}$$

The intention is to demonstrate that $Spec \approx_L P$ by deriving the sequent

$$\vdash \mathbf{tt} \rhd \tau.Spec = X_0,$$

We follow the construction prescribed by the proof of completeness and transform $\tau.Spec$ into a saturared standard form, Y_0. Let $D_2 =$

$$\left\{\begin{array}{lcl}
Y_0 & \Longleftarrow & \tau.Y_1 + i?w.Y_2(0,w) \\
Y_1 & \Longleftarrow & i?w.Y_2(0,w) \\
Y_2 & \Longleftarrow & \lambda yy'.[o!(y+y').Y_4(y+y') + i?w.Y_3(y+y',w)] \\
Y_3 & \Longleftarrow & \lambda yy'.o!y.Y_2(y+y') \\
Y_4 & \Longleftarrow & \lambda y.i?w.Y_2(y,w)
\end{array}\right\}$$

We can now apply Theorem 6.4.5 to obtain the common declaration, E. We first list the invariants $\{b_{ij}\}_{I\times J}$ we require. The majority of these are $\mathbf{ff}$, those that aren't are given below:

$$\begin{array}{lcl}
b_{00} & = & \mathbf{tt} \\
b_{11} & = & x = 0 \\
b_{22} & = & x = y+y' \\
b_{14} & = & x = y \\
b_{32} & = & x = y+y' \\
b_{43} & = & x = y \wedge x' = y'
\end{array}$$

We can now write down the declaration $E = \{Z_{ij}\}$, omitting the definition of all the Z_{ij} such that $b_{ij} = \mathbf{ff}$.

$$\left\{\begin{array}{lcl}
Z_{00} & \Longleftarrow & \tau.Z_{11}(0) + i?w.Z_{22}(0+w,0,w) \\
Z_{11} & \Longleftarrow & \lambda x.b_{11} \rightarrow i?w.Z_{22}(x+w,0,w) \\
Z_{22} & \Longleftarrow & \lambda xyy'.b_{22} \rightarrow [o!x.Z_{14}(x,y+y') + \tau.Z_{32}(x,y,y') + i?w.Z_{43}(x,w,y+y',w))] \\
Z_{14} & \Longleftarrow & \lambda xy.b_{14} \rightarrow i?w.Z_{22}(x+w,y,w) \\
Z_{32} & \Longleftarrow & \lambda xyy'.b_{32} \rightarrow [o!x.Z_{14}(x,y+y') + i?w.Z_{43}(x,w,y+y',w)] \\
Z_{43} & \Longleftarrow & \lambda xx'yy'.b_{43} \rightarrow o!x.Z_{22}(x+x',y,y')
\end{array}\right\}$$

This declaration indicates that, to apply UFI-Inv to prove $\vdash_{D_2 \cup E} Y_0 = Z_{00}$ we would use the terms

$$\begin{array}{lcl}
f_{00} & \equiv & Y_0 \\
f_{11} & \equiv & \lambda x.Y_1 \\
f_{22} & \equiv & \lambda xyy'.\tau.Y_2(y,y') \\
f_{14} & \equiv & \lambda xy.Y_4(y) \\
f_{32} & \equiv & \lambda xyy'.Y_2(y,y') \\
f_{43} & \equiv & \lambda xx'yy'.Y_3(y,y')
\end{array}$$

We leave it to the reader to check that the loop invariants are maintained and that the hypotheses of the UFI-Inv can be established. Proving $\vdash_{D_1 \cup E} X_1(0) = Z_{11}(0)$ is much easier and transitivity and dec-I finish the derivation.

Chapter 7

Conclusion

7.1 Summary

Whilst it is acknowledged that proving properties of value-passing systems, and indeed developing proof tools for such tasks is, in general, a difficult task, the purpose of this thesis was to demonstrate that partial solutions to this problem can be readily achieved. We can extract requisite properties of the data which guarantee certain behavioural properties of processes and can then analyse and reason about value-passing processes structurally. The manner in which we extract and model these properties is through the mechanism of symbolic graphs, or by syntactically studying open terms. The use of symbolic semantics on open terms has been successful in various settings already, [40, 41, 43, 64]. The present text has strengthened this corpus of work considerably and, at the same time, provided some independently interesting results regarding the semantics of value-passing calculi. At all stages of the thesis both the late and early semantics of value-passing processes are considered.

A novel equivalence relation, based on barbed bisimulation, for Prasad's calculus of broadcasting systems, CBS, was discovered in Chapter 3. We then characterised this new relation algebraically by developing a sound and relatively complete equational proof system for various classes of finite CBS processes. This characterisation relied heavily upon the use of a new, symbolic, semantics for CBS. These new semantics involve a modification of the semantics envisaged by Hennessy and Lin, [40], in that pattern-matching of values on input prefixes need to be catered for. The use of, and generalisations of, these semantics are investigated further in [77]. The characterisations of Chapter 3 lay the foundation for tackling the open problem [86] of finding an axiomatic description of weak bisimulation equivalence in CBS. In Chapter 4 we verified, again using the barbed bisimulation approach, that Prasad's weak equivalence is indeed suitable for this language and proceeded to utilise the symbolic semantics developed in the previous chapter in order to provide the sought after axiomatisation.

An entirely different problem was tackled in Chapter 5, that of model checking for value-passing processes. A generalisation of the modal μ-calculus suitable for specifying properties not only of the ability to perform actions but also of the data associated with actions was proposed. This generalised specification logic is a direct extension of the logic presented in [43] and similar logics have been developed, independently, by [38, 27, 3]. We showed that, by interpreting the logic using concreted symbolic graphs we were able to generalise Winskel's tag set method, [107], for completing proof tableaux in a proof system developed to make use of the underlying symbolic graph models. We discovered that, even for finite symbolic graphs, the specification logic is too expressive for us to show any form of completeness result using our simple tableaux system. Two different sub-logics provide enough restrictions, however, for us to achieve relative completeness results for each. For one of these, a symbolic interpretation of formula in the spirit of [27] was necessary.

Finally, we considered the technique of unique fixpoint induction as a proof method for value-passing processes. Work here naturally extends the work of [41] by providing proof systems (relative to a language of boolean expressions) for recursively defined processes. It also extends the recent work [42, 64] by

- Considering a larger class of recursively defined processes, and
- Providing a characterisation of observational equivalence as well as strong bisimulation equivalence for this class of processes.

We finished this chapter with a discussion on the relationship between parameterisation and parallel composition of recursively defined value-passing processes.

Therefore, we can deem this thesis to be successful in that it provides an original analysis of various problems using the symbolic approach; in each case results suggest that the reasoning required in each of these scenario can be arranged so that reasoning about processes is reducible, in an automatic manner, to reasoning about data. Furthermore, these studies indicate what sort of properties about data one might be expected to prove. For instance, Chapters 5 and 6 both require the use of properties of data expressed in a language of fixpoints.

7.2 Related work

7.2.1 Symbolic bisimulation vs. Abstract interpretation

An alternative approach to verification for value-passing processes, quite similar in spirit to our own, is that of abstract interpretation. The technique of abstract interpretation has been used successfully for many years now for the analysis of sequential programs [55, 1]. More recently, this approach has been applied to reactive systems and, in particular, labelled transition systems [9, 23, 28, 26]. The idea being that enough structure of some transition systems can sometimes be preserved after abstracting, or collecting together, certain labels. By *enough* structure we mean a level of structural information which still allows the property which one is interested in to be verified. The clear benefits of this approach are that the transition models become smaller and verification becomes more feasible. This approach was followed by Cleaveland and Riely, [26], in order to verify testing equivalences in value-passing languages. Abstract interpretation of value domains induce abstract transition models for processes. In some cases, if a coarse enough abstraction is used, then the transition systems can be reduced from infinite concrete systems down to finite abstract systems.

This is an elegant approach to verification of equivalences for value-passing languages. The abstract transition systems can be considerably smaller than the actual transition systems, thereby improving effectiveness and efficiency. However the approach is not without its drawbacks. There can be a considerable amount of work involved in actually creating an abstraction of the data language. Each value must be given an abstract value and each function in the signature of data expressions must be interpreted abstractly in a safe manner. One might envisage libraries of standard abstractions being created to avoid much of the effort here. Such considerations aside there is still the initial choice of which abstract values to use, which is largely determined by educated guesswork. One must find an interpretation which is coarse enough to reduce your transition system to a manageable size yet at the same time provide, safe, accurate results. Thus, much of the efficiency and practicality gained through the abstract interpretation approach may be lost in the search for a good abstraction.

In contrast to this we see that symbolic graphs provide an easily obtained form of abstract interpretation. The expression x in the label $c!x$ appearing on an arc of a symbolic graph can be viewed, roughly, as the range of values that x could be instantiated to. Recall arcs of the graph are guarded by boolean triggers which may restrict the range of values which x can take. In a suitable sense, sets of concrete values can be seen as abstract values. The difference in approach lies in the interpretation of the functions in the data signature. Our approach allows for a kind of *precise* interpretation only; interpretation of functions is determined by the abstraction on values. So the abstract meaning f_A, of a function f of arity one, is defined as

$$f_A(V) = \{f(v) \mid v \in V\}$$

where V, being an abstract value, is a set of concrete values from Val. Thus we are unable to reap some of the benefits of general abstraction. For example, if we wish to demonstrate deadlock freedom of the process $p(x)$, where

$$p \Longleftarrow \lambda y.c!y.p(y+1)$$

then the symbolic semantics induces *abstract values* by considering the concrete values that x may take. Initially, x could be any value, which we represent by the set Val. The second output from $p(x)$ would need to carry the abstract value $Val + 1$. Our semantics demand that $Val + 1$ be the set $Val \setminus \{0\}$ and subsequently that $(Val \setminus \{0\}) + 1$ is $Val \setminus \{0,1\}$ and so on. Using abstract interpretation proper, $p(x)$ can be safely abstracted to the process

$$p_A \Longleftarrow c!0.p_A$$

where 0 is now the only abstract value. This is a situation where symbolic graphs fail, although the work on symbolic graphs with assignment of [65, 78] and that of Chapter 6 provide more success. In fact, the only regular value-passing processes of value-passing CCS, say, which cannot be modelled finitely by a symbolic graph with assignment are those with unguarded recursions. Another benefit of the symbolic approach is the fact that symbolic graphs are constructed automatically. The abstractions involved are entirely syntax driven, which obviates the need for extensive searching for a good abstraction.

7.2.2 μCRL

The work of Groote and Ponse, [34, 35, 37], amongst others, adopts a slightly different world view to the symbolic approach to value-passing. It is our opinion that properties of data and the properties of the process behaviour should be made distinct, or at least, the properties of the process behaviour which depend on aspects of the data should be extracted from the overall process description. For this reason we work parametrically with respect to the data language. The language μCRL was designed with a different methodology in mind. The intention there is to integrate as much as possible the description of the data language and the process language. The programmer defines which data and data expressions can be used by means of algebraic specification as in the ISO-standard LOTOS specification language [53], in contrast though, data live as first class citizens in μCRL. Thus proofs of properties of μCRL processes are not relative to any unspecified data language but are total verifications. Consequently, the lion's share of the work involved in both specifying processes and verification is, to a large extent, due to data. This situation is acknowledged in [37] which catalogues some widely applicable data specifications and inductive proof techniques. There has been commendable success, with realistic examples, using the μCRL language as a verification tool [33, 57, 36]. However, the theory underpinning the approach is still in development. Groote and Ponse provide sound proof systems for establishing bisimilarity between μCRL processes but it is unclear how powerful these proof systems actually are. An anaylsis of the dependency of a process on its data component is made difficult by integrating proof systems for data and for process terms. The actual structure of a process is not made explicit and is obscured by syntax. Whereas the symbolic graph models provide such structural information, only retaining syntactic descriptions of data.

7.3 Directions for future work

The work in this thesis raises several issues that deserve further attention.

- Having developed a notion of morphism for symbolic graphs it would be interesting to consider if symbolic bisimulation upon saturated graphs could be characterised as by spans of *open* morphisms in the style of Joyal, Nielsen, and Winskel, [56]. This programme would involve identifying a suitable notion of a *path* of observations, akin to a trace, for symbolic graphs.

- The relationship between abstract interpretation and the symbolic approach is not well understood. For instance, could we allow for some more general form of abstraction by relaxing the conditions of symbolic graphs somehow? Such analysis could improve the efficiency of symbolic reasoning greatly. A formal description of symbolic graphs as abstractly interpreted transition systems would be desirable.

- The proof systems of Chapters 3 and 4 deal with finite CBS only. There is no evident reason why the unique fixpoint induction proof method would not prove useful for characterising equivalence of regular CBS terms. We anticipate that a programme analagous to that of Chapter 6 could be followed without great difficulty.

- The work of Chapter 5 attacks the model checking problem for symbolic graphs. Relative completeness results are obtained for finite symbolic graphs only. It should be clear from Chapter 6 that finite symbolic graphs are somewhat limited in their scope for modelling value-passing processes and that using symbolic graphs with assignment yields finite models in many more cases. For this reason then, it would be desirable to investigate the model checking problem for symbolic graphs with assignment. The modality rules in the proof system are easily adapted to this setting, for example $\langle c! \rangle$ would become

  $$\frac{B \vdash t'\theta : F[e\theta/x]}{B \wedge b \vdash t : \langle c!x \rangle F} \quad t \overset{b,\theta,c!e}{\longmapsto} t'.$$

  The difficulties lie in using the tag sets effectively. A node of a symbolic graph represents an open process term, but a node of a symbolic graph with assignment has no such counterpart. It has to represent an open term possibly with many different substitutions of the free variables with data expressions. Thus the semantics of a tag would necessarily need to account for this fact. Beyond this problem though, the approach of Chapter 5 ought to follow through without further difficulty.

  Another logical extension to the work would be to consider incorporating some kind of inductive reasoning on data in order to obtain completeness results for the logic with least fixpoints and arbitrary parameters. Based on a technique from [5, 14] a global proof rule following this idea is included in the proof system of [38]. The side-condition of this rule involves establishing a well-founded order on the value domain.

- The shortfall in the theory of Chapter 6 concerns the issue of unguarded recursions or, in the weak case, recursions with τ loops. In the pure case it was shown in [74] that unguardedness and τ loops could be effectively removed from declarations. The main rule for achieving this,

  $$\text{If } \; X \Longleftarrow X + p \; \text{ then } \; \vdash X = p,$$

  relies upon the fact that the unguarded occurrence of X creates no *new* actions. That is, $X + p$ can still only perform the actions that p could perform because of idempotence of $+$. In the value-passing setting the situation differs because of parameterisation. Each unguarded identifier's body may become instantiated at a different value after each unfolding. For example, the process

$$X \Longleftarrow \lambda x.(X(x+1) + a!x.\mathbf{O}),$$

when instantiated at 0 has an infinitely branching symbolic graph. Guarded recursions always have finitely branching symbolic graphs so it is clear that X cannot be reduced to a guarded declaration. Two immediate approaches spring to mind: restrict the form of the parameters allowed in unguarded recursions or introduce some kind of inductive reasoning on data to allow proofs between unguarded recursive declarations. We can predict success for the former of these approaches in the case where parameters are restricted to be variables alone. Such a restriction is in place in the π-calculus where recent work by Lin, [62], shows that unguardedness can be tackled by saturating declarations with permutations of variables. The latter approach is somewhat more uncertain. The principle that one might use is that if, given two declarations whose guarded parts are provably equal, we know that the unguarded occurrences in each side give rise to the same instantiations of the guarded parts of the declarations, then we can deem the two declarations to be equal. Roughly, this might look like: if

$$X \Longleftarrow \lambda \bar{x}.X(e_1) + \ldots + X(e_n) + t$$

and

$$Y \Longleftarrow \lambda \bar{x}.Y(e'_1) + \ldots + Y(e'_m) + u$$

with t and u guarded, then, writing $X^g \Longleftarrow \lambda \bar{x}.t[X^g/X]$ and similarly for Y^g,

$$\frac{\vdash X^g = Y^g}{\vdash X = Y}$$

if $e_{i_1} \circ e_{i_2} \circ \ldots \circ e_{i_k} = e'_{j_1} \circ e'_{j_2} \circ \ldots \circ e'_{j_l}$ for any sequences of expressions. A good account of unguardedness, and more importantly, τ loops, would finally establish the proof systems for value-passing processes on the same footing as the proof systems for pure processes - modulo the data.

- In order to obtain relative completeness results in Chapters 5 and 6 we needed to consider a logic of fixpoints over boolean predicates. We found this necessary to describe the conditions on data required to guarantee certain properties of the processes. This is not an entirely satisfactory situation as first-order logic with fixpoints is a rather hefty logic. In many cases it will be that these fixpoints are actually expressible in the base languange *BoolExp*; however, in general, we cannot guarantee this. An interesting topic of research would be to attempt to classify data languages for which these fixpoints have first-order solutions, or are even expressible in some decidable second-order theory.

- We have referred to the use of a symbolic approach to the π-calculus, [63, 64, 62] at various points in this thesis. We can think of such work as an instance of the general symbolic technique where the values of the data language and the channel names coincide. There are further technicalities involved but this is the case in principle. Similarly, adopting the symbolic techinque in other areas might also prove useful. For instance, work on utilising symbolic bisimulations for a theory of timed processes has already begun, [13]. Recent work on location equivalences has recourse to a symbolic semantics, [49]. Such an approach may even prove useful for considering the theory of higher-order process algebras such as Chocs [106], Facile [30] or CML based languages [93, 29, 54].

Bibliography

[1] S. Abramsky and C. Hankin. *Abstract interpretation of declarative languages*. Ellis Horwood, 1987. (p 135)

[2] P. Aczel. *Non-well-founded Sets*, volume 14 of *CSLI Lecture Notes*. Stanford University, 1988. (p 3)

[3] R. Amadio and M. Dam. Toward a modal theory of types for the π-calculus. In *Proc. Formal Techniques in Real Time and Fault Tolerant Systems, Uppsala 96*, volume 1135 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996. (p 7)

[4] H. Andersen, C. Stirling, and G. Winskel. A compositional proof system for the modal μ-calculus. In *Proceedings* 9^{th} *Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1994. (p 7)

[5] H.R. Andersen. *Verification of Temporal Properties of Concurrent systems*. PhD thesis, Department of Computer Science, Aarhus University, Denmark, June 1993. (p 137)

[6] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990. (pp 1, 10, 128)

[7] H. Bekič. Towards a mathematical theory of processes. Technical Report TR25.125, IBM Laboratory, Vienna, 1971. This document also appeared in [8]. (p 1)

[8] H. Bekič. Programming languages and their definition Springer Lecture Notes in Computer Science Volume 177, 1984. (p 139)

[9] S. Bensalem, A. Bouajjani, C. Loiseaux, and J. Sifakis. Property-preserving simulations. In *Proceedings of the Workshop on Computer-Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 260–273. Springer-Verlag, 1992. (p 135)

[10] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985. (p 1)

[11] G. Berry and L. Cosserat. The ESTEREL synchronous programming language and its mathematical semantics. In S.D. Brookes, A.W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 389–448. Springer-Verlag, 1984. (p 6)

[12] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19, 1992. (p 6)

[13] M. Boreale. Symbolic bisimulation for timed processes. In M. Wirsing and M. Nivat, editors, *Proc. 5th Conference on Algebraic Methodologies and Software Technology (AMAST'96)*, volume 1101 of *Lecture Notes in Computer Science*, pages 321–335. Springer-Verlag, 1996. (p 138)

[14] J. Bradfield and C. Stirling. Local model checking for infinite state spaces. *Theoretical Computer Science*, 96:157–174, 1992. (pp 7, 74, 137)

[15] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984. (p 1)

[16] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986. (p 7)

[17] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992. (pp 3, 7)

[18] N. Carriero and D. Gelernter. LINDA in context. *Communications of the ACM*, 32(4):444–458, 1989. (p 6)

[19] S. Christensen, Y. Hirshfield, and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In E. Best, editor, *Proceedings CONCUR 93,* Hildesheim, volume 715 of *Lecture Notes in Computer Science*, pages 143–157. Springer-Verlag, 1993. (p 4)

[20] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121(2):143–148, 1995. (p 4)

[21] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Proceedings of the Workshop on Logic of Programs,* Yorktown Heights, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981. (p 7)

[22] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986. (p 7)

[23] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. In *Conference Record of the Nineteenth Annual ACM Symposium on Principles of Programming Languages*, pages 343–354, 1992. (pp 3, 7, 135)

[24] R. Cleaveland. Tableau-based model checking in the propositional μ-calculus. *Acta Informatica*, 27:725–747, 1990. (p 7)

[25] R. Cleaveland, M. Dreimüller, and B. Steffen. Faster model checking for the modal μ-calculus. In *CAV'92*, volume 663 of *Lecture Notes in Computer Science*, pages 383–394. Springer-Verlag, 1993. (p 7)

[26] R. Cleaveland and J. Riely. Testing-based abstractions for value-passing systems. In B. Jonsson and J. Parrow, editors, *Proceedings CONCUR 94,* Uppsala, volume 836 of *Lecture Notes in Computer Science*, pages 417–432. Springer-Verlag, 1994. (p 135)

[27] M. Dam. Model checking mobile processes. In E. Best, editor, *Proceedings CONCUR 93,* Hildesheim, volume 715 of *Lecture Notes in Computer Science*, pages 22–36. Springer-Verlag, 1993. (pp 7, 81, 134)

[28] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretations of reactive systems: Abstractions preserving $\forall CTL^*$, $\exists CTL^*$, and CTL^*. In *Proceedings PROCOMET'94, IFIP Transactions*. North-Holland/Elsevier, 1994. (p 135)

[29] W. Ferreira, M. Hennessy, and A.S.A Jeffrey. A theory of weak bisimulation for core CML. In *Proc. ACM SIGPLAN Int. Conf. Functional Programming*. ACM Press, 1996. (p 138)

[30] A. Giacalone, P. Mishra, and S. Prasad. Facile: A symmetric integration of concurrent and functional programming. In *Proceedings TAPSOFT89 conference*, volume 352 of *Lecture Notes in Computer Science*, pages 184–209. Springer-Verlag, 1989. (p 138)

[31] A. Gordon. *Functional Programming and Input/Output*. Distinguished dissertations in computer science. Cambridge University Press, 1994. (p 3)

[32] A. Gordon. Bisimilarity as a theory of functional programming. Mini-course BRICS-NS-95-3, BRICS, Department of Computer Science, Aarhus University, 1995. (p 3)

[33] J.F. Groote and H. Korver. A correctness proof of the bakery protocol in μCRL. Logic Group Preprint Series 80, Dept. of Philosophy, Utrecht University, October 1992. (p 136)

[34] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. Report CS-R9076, CWI, Amsterdam, 1990. (p 136)

[35] J.F. Groote and A. Ponse. Proof theory for μCRL. Report CS-R9138, CWI, Amsterdam, August 1991. (p 136)

[36] J.F. Groote and J.G. Springintveld. Algebraic verification of a distributed summation algorithm. Report CS-R9640, CWI, 1996. (p 136)

[37] J.F. Groote and J.J. van Wamel. Algebraic data types and induction in μCRL. Report P9409, University of Amsterdam, 1994. (p 136)

[38] D. Gurov, S. Berezin, and B. Kapron. A modal μ-calculus and a proof system for value-passing processes. In *Proc. Infinity Workshop on Verification of Infinite State Systems, Pisa*, pages 149–163, 1996. (pp 7, 134, 137)

[39] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, Cambridge, Massachusetts, 1988. (p 2)

[40] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353–389, 1995. (pp 3, 5, 17, 24, 38, 115, 134)

[41] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. *Formal Aspects of Computer Science*, pages 379–407, 1996. (pp 5, 8, 9, 34, 36, 37, 38, 39, 102, 103, 104, 105, 117, 128, 134, 135)

[42] M. Hennessy and H. Lin. Unique fixpoint induction for message-passing process calculi. In *Proceedings of CATS97, Computing:Australian Theory Symposium*, 1997. To appear. (pp 8, 9, 103, 104, 105, 107, 108, 109, 110, 117, 126, 128, 135)

[43] M. Hennessy and H. Liu. A modal logic for message passing processes. *Acta Informatica*, 32:375–393, 1995. (pp 5, 9, 74, 75, 76, 80, 81, 85, 90, 98, 134)

[44] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985. (pp 1, 2, 20, 74)

[45] M. Hennessy and G.D. Plotkin. A term model for CCS. In P. Dembiński, editor, 9^{th} *Symposium on Mathematical Foundations of Computer Science*, volume 88 of *Lecture Notes in Computer Science*, pages 261–274. Springer-Verlag, 1980. (p 34)

[46] M. Hennessy and J. Rathke. Bisimulations for a calculus of broadcasting systems. In I. Lee and S. Smolka, editors, *Proceedings CONCUR 95,* Philadelphia, volume 962 of *Lecture Notes in Computer Science*, pages 486–500. Springer-Verlag, 1995. (p ii)

[47] M. Hennessy and J. Rathke. Strong bisimulations for a calculus of broadcasting systems. Computer Science Report 1/95, University of Sussex, 1995. (p ii)

[48] M. Hennessy and J. Rathke. Weak bisimulations for a calculus of broadcasting systems. Computer Science Report 3/95, University of Sussex, 1995. (p ii)

[49] M. Hennessy and J. Riely. Distributed processes and location failures. Computer science report, University of Sussex, 1997. To appear. (p 138)

[50] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978. (p 1)

[51] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, Englewood Cliffs, 1985. (pp 1, 2, 8, 10, 128)

[52] A. Ingólfsdóttir. *Semantic Models for Communicating Process with Value-Passing*. PhD thesis, University of Sussex, September 1993. (p 3)

[53] ISO. *Information processing systems – open systems interconnection – LOTOS – a formal description technique based on the temporal ordering of observational behaviour* ISO/TC97/SC21/N DIS8807, 1987. (p 136)

[54] A.S.A. Jeffrey. A fully abstract semantics for a concurrent functional language with monadic types. In *Proceedings* 10^{th} *Annual Symposium on Logic in Computer Science,* San Diego, pages 255–264. IEEE Computer Society Press, 1994. (p 138)

[55] N.D. Jones and F. Nielson. Abstract interpretation: a semantics based tool for program analysis. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 4*, pages 527–636. Oxford University Press, 1995. (p 135)

[56] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation and open maps. In *Proceedings* 8^{th} *Annual Symposium on Logic in Computer Science*, pages 418–427. IEEE Computer Society Press, 1993. (pp 20, 137)

[57] G. Kamsteeg. A formal verification of the alternating bit protocol in μCRL. Report CS-R9337, Leiden University, 1993. (p 136)

[58] R.M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976. (p 2)

[59] K.G.Larsen. Proof systems for Hennessy-Milner logic with recursion. *Lecture Notes in Computer Science*, page 299, 1988. (pp 7, 74)

[60] J. Kleist and M. Hansen. Process calculi with asynchronous communication. Report, Aarhus University, 1994. (p 6)

[61] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–353, 1983. (pp 7, 10, 74, 80, 94)

[62] H. Lin. Complete proof systems for weak bisimulation equivalences in the π-calculus with recursion. Forthcoming technical report at University of Sussex. (pp 8, 138)

[63] H. Lin. Symbolic bisimulations and proof systems for the π-calculus. Computer Science Report 7/94, University of Sussex, 1994. (p 138)

[64] H. Lin. Unique fixpoint induction for mobile processes. In I. Lee and S. Smolka, editors, *Proceedings CONCUR 95,* Philadelphia, volume 962 of *Lecture Notes in Computer Science*, pages 88–102. Springer-Verlag, 1995. (pp 8, 103, 108, 138)

[65] H. Lin. Symbolic graphs with assignment. In U. Montanari and V.Sassone, editors, *Proceedings CONCUR 96,* Pisa, volume 1119 of *Lecture Notes in Computer Science*, pages 50–65. Springer-Verlag, 1996. (pp 24, 25, 125, 136)

[66] D. M. Park Luckham, D.C. and M. S. Paterson. On formalized computer programs. *J. Computer System Sciences*, (4):220–249, 1970. (p 78)

[67] Z Manna and A Pnueli. Formalization of properties of recursively defined functions. In *Conference Record of ACM Symposium on Theory of Computing*, pages 201–210, Marina del Rey, California, 5–7 May 1969. (p 78)

[68] R.M. Metcalfe and D.R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7), 1976. (p 6)

[69] G. Milne and R. Milner. Concurrent processes and their syntax. *Journal of the ACM*, 26(2):302–321, 1979. (p 1)

[70] R. Milner. Processes: A mathematical model of computing agents. In H.E. Rose and J.C. Shepherdson, editors, *Proceedings Logic Colloquium 1973*, pages 158–173. North-Holland, 1973. (p 1)

[71] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980. (p 1)

[72] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984. (pp 7, 102)

[73] R. Milner. A complete axiomatisation for observational congruence of finite-state behaviours. Technical Report ECS-LFCS-86-8, Department of Computer Science, University of Edinburgh, 1986. (pp 7, 102, 110, 119)

[74] R. Milner. *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs, 1989. (pp 1, 2, 8, 10, 27, 30, 40, 58, 59, 62, 102, 115, 116, 128, 137)

[75] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992. (pp 4, 6, 20, 21)

[76] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings* 19th *ICALP,* Vienna, volume 623 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992. (p 6)

[77] P. Paczkowski. Verifying CBS processes using symbolic semantics. In *Proc. 7th Nordic Workshop on Programming Theory*, pages 310–323, 1995. Report 86, Programming Methodology Group, Göteborgs University and Chalmers University of Technology. (p 134)

[78] P. Paczkowski. Characterising bisimilarity of value-passing parameterised processes. In *Proc. Infinity Workshop on Verification of Infinite State Systems*, pages 47–55, 1996. (pp 24, 25, 125)

[79] D. M. Park. Fixpoint induction and proofs of program properties. *Machine Intelligence*, (5):59–78, 1970. (p 78)

[80] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5th *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981. (pp 1, 2, 4)

[81] J. Parrow and D. Sangiorgi. Algebraic theories for value-passing calculi. *Information and Computation*, pages 174–197, 1995. (p 126)

[82] C.A. Petri. Kommunikation mit automaten. Schriften des IIM 2, Institut für Instrumentelle Mathematik, Bonn, 1962. (p 1)

[83] C.A. Petri. Concurrency theory. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986, Part I, Petri Nets: Central Models and Their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 4–24. Springer-Verlag, 1987. (p 1)

[84] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In Wilfried Brauer, editor, *Proceedings 12th ICALP,* Nafplion, volume 194 of *Lecture Notes in Computer Science*, pages 15–32. Springer-Verlag, July 1985. (p 6)

[85] K.V.S. Prasad. A calculus of broadcast systems. In *TAPSOFT 91 Volume 1: CAAP*. Springer Verlag, 1991. (pp 6, 26)

[86] K.V.S. Prasad. A calculus of value broadcasts. Technical report, Dept. of Computer Science, Chalmers, 1992. (pp 6, 9, 26, 27, 30, 53, 56, 57, 134)

[87] K.V.S. Prasad. Programming with broadcasts. In E. Best, editor, *Proceedings CONCUR 93,* Hildesheim, volume 715 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993. (pp 6, 26)

[88] V. Pratt. A decidable μ-calculus. In *Proceedings 22nd Annual ACM Symposium on Foundations of Computer Science*, pages 421–427, 1981. (pp 7, 10, 74)

[89] J.P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the fifth international symposium in programming*, 1981. (p 7)

[90] J. Rathke. Unique fixpoint induction for value-passing processes. In *Proc.* 12th *Annual Symposium on Logic in Computer Science, Warsaw*. IEEE Computer Society Press, 1997. To appear. (p ii)

[91] J. Rathke and M. Hennessy. Local model checking for a value-based modal μ-calculus. Technical Report 05/96, University of Sussex, 1996. (p ii)

[92] W. Reisig. *Petri nets – an introduction.* EATCS Monographs on Theoretical Computer Science, Volume 4. Springer-Verlag, 1985. (p 1)

[93] J. Reppy. *Higher-Order Concurrency.* PhD thesis, Cornell University, June 1992. Technical Report TR 92-1285. (p 138)

[94] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1993. (pp 6, 29, 58, 59)

[95] D. Sangiorgi and R. Milner. On the problem of 'weak bisimulation up to'. In W.R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1992. (p 116)

[96] M.Z. Schreiber. *Value-passing Process Calculi as a Formal Method.* PhD thesis, Imperial College, 1994. (p 24)

[97] P. Stevens, 1995. Talk at ESPRIT BRA Concur2 project workshop, Cannes. (p 129)

[98] C. Stirling. Modal logics for communicating systems. *Theoretical Computer Science*, pages 311–347, 1987. (p 74)

[99] C. Stirling. Modal and temporal logics. In S. Abramsky, D. Gabbay, and T.Maibaum, editors, *Handbook of Logic in Computer Science, Vol I.* Oxford University Press, 1990. (pp 10, 74)

[100] C. Stirling. Modal and temporal logics for processes, 1993. Notes for Summer School in Logic Methods in Concurreny, Aarhus University. (p 20)

[101] C. Stirling. Local model checking games. In I. Lee and S. Smolka, editors, *Proceedings CONCUR 95,* Philadelphia, volume 962 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1995. (p 7)

[102] C. Stirling. Decidability of bisimulation equivalence for normed pushdown processes. In U. Montanari and V. Sassone, editors, *Proceedings CONCUR 96,* Pisa, volume 1119 of *Lecture Notes in Computer Science*, pages 217–232. Springer-Verlag, 1996. (p 4)

[103] C. Stirling and D. Walker. Local model checking in the modal μ-calculus. *Theoretical Computer Science*, 89:161–177, 1991. (pp 7, 74, 75)

[104] A. Stoughton. Substitution revisited. *Theoretical Computer Science*, 59:317–325, 1988. (p 103)

[105] W. Thomas. On the Ehrenfeucht-Fraisse game in theoretical computer science. In *Proceedings TAPSOFT 93*, volume 668 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993. (p 20)

[106] B. Thomsen. *Calculi for Higher-Order Communicating Systems*. PhD thesis, University of London, 1990. (p 138)

[107] G. Winskel. A note on model checking the modal ν-calculus. *Theoretical Computer Science*, 83:157–167, 1991. (pp 7, 9, 74, 75, 81, 94, 134)

[108] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 4*, pages 1–148. Oxford University Press, 1995. (p 11)

Index