Games for Logic and Concurrency

Julian Gutierrez
J.E.Gutierrez@ed.ac.uk
Laboratory for Foundations of Computer Science
School of Informatics. University of Edinburgh
United Kingdom

Supervisors:
Dr. Julian Bradfield and Dr. Ian Stark

Contents

1 Project aims and objectives

1.1 Project aim

Concurrency theory is concerned with the logical and mathematical study of parallel processes. These systems can be analysed by studying issues related to logical formalisms employed to specify and verify their properties as well as the models used to represent their structure and behaviour. We shall address these issues in this dissertation. *The aim of this project is to formulate techniques based on games to study the logical specification and verification of parallel processes and their mathematical models.*

Our main goal is to define game procedures for three major problems related to the analysis of concurrent systems, namely, the satisfiability and model checking problems for temporal logics, and the equivalence checking problem for different models of concurrency. We shall do this by studying a fixpoint modal logic, which extends the modal mu-calculus and captures aspects of concurrency and locality of its underlying models, as well as by studying equivalences between the events in such models.

1.2 Project specific objectives

- *To define a fixpoint modal logic that extends the modal mu-calculus and captures aspects of concurrency and locality of parallel processes.* This formalism, whose natural models are true concurrency structures, will be based on the introduction of logical operators sensitive to local information about concurrency of independent parallel components. The basic language will be enriched with fixpoints to be able to study both linear-time and branching-time temporal properties of concurrent systems.

- *To formulate a class of games for satisfiability and completeness for fixpoint logics.* These games will be based on the procedures developed by Stirling and Lange in [15]. The games presented in [15] address the satisfiability problem of a number of linear-time and branching-time temporal logics, and allow for an elegant extraction of their axiomatizations. Our approach will be to formulate a concurrent or asynchronous extension of these games to try to overcome the problems that prevent their use with arbitrary fixpoint logics for concurrent systems.

- *To develop asynchronous or concurrent games for model checking fixpoint modal logics.* This piece of work was inspired by the fact that most model checking games for modal and temporal logics are essentially sequential (cf. [14, 16]). However, in other contexts, such as, game semantics of programming languages or game models of reactive systems (e.g., games on graphs), concurrent games are an alternative approach. In these scenarios, concurrent games try to capture information about independence in the systems they represent. In our case, a logical context, concurrent or asynchronous games should use as much information as possible about concurrency in the models so as to verify their properties.

- *To study a game-theoretic approach to the problem of equivalence checking for models of concurrency.* Another important problem when studying parallel systems is that of characterising their mathematical models. One possibility to do this is by defining equivalences or relationships between the structures used to give denotation to temporal formulae and semantics to process algebras. Since these structures are usually models of concurrency, such as, Labelled transition systems or Petri nets, traditional approaches to tackle this problem have focused on defining games to identify bisimilarities between particular instances of models, e.g., bisimilarities between two Labelled transition systems. We, instead, are interested

in defining more abstract relationships that allow us to compare models of concurrency without looking at their actual structures, but rather the structure of their events, which we could assume partially ordered. We will study a game-theoretic characterisation of equivalences between events of models of concurrency by using Galois connections, so as to find correspondences between the structures at hand, and therefore, the models taken into consideration.

2 Project description

2.1 Introduction and motivation

Games have proved to be very useful in Computer Science in several areas and for different purposes. Some remarkable examples are their usage to give fully abstract denotational semantics to programming languages, to model the behaviour of multi-agent systems, to study equivalences between particular mathematical and computational structures, to give semantics to logical languages, or to verify temporal properties of concurrent systems, among others.

Most of these games have been defined to be played sequentially, i.e., it is not the case that two players or agents make a move at the same time because either they take turns or alternate their actions in the game. This setting has changed in the last few years in order to address some problems in concurrency. Abramsky and Mellies defined various classes of concurrent and asynchronous games [1, 17], in which players can have strategies where several moves can be made at once and at the same time. Their main application is in the area of Formal Programming Language Semantics. Also, Henzinger et al. have used concurrent game structures, which are graphs where several players can interact synchronously, to model the behaviour of reactive systems regarded as open ones. These concurrent games, and also variants of them, have been used to study ω-regular properties of reactive systems as well as to give semantics to a number of temporal logics for open systems [2, 8, 9].

However, much work still remains to be done in other research areas of Logic and Concurrency. We are interested in games for fixpoint modal and temporal logics as well as in games for checking equivalences between different mathematical structures representing the behaviour of concurrent systems. In these kinds of games the idea of independence as some sort of concurrency or asynchrony has not yet been fully investigated. We shall do so in this thesis. The main motivations are the need for techniques to overcome some of the problems that appear when using traditional sequential games for verification of modal and temporal logics (cf. [14, 15, 16]), and to give a game-theoretic characterisation to the problem of defining equivalences between different models of (non-interleaving) concurrency, something that, so far, has had a prominently purely categorical flavour in the true concurrency context [26].

Therefore, this project will aim to investigate how sequential games for satisfiability, completeness and model checking of fixpoint modal logics can be defined in an asynchronous or concurrent setting. The games to be studied are those developed by Lange and Stirling for several linear-time and branching-time modal and temporal logics [14, 28]. With the purpose of having a better understanding of the relationships between the notion of concurrency in games for verification and that in the logics for specification, logical operators for expressing independence in a specification language will also be introduced. We will define a fixpoint modal logic, sensitive to aspects of concurrency and locality of parallel processes, that extends the modal mu-calculus and whose natural models are true concurrency structures. After that, a partially ordered representation of the events in these structures will be used to define games that deal with equivalences between models of concurrency.

In summary, this Ph.D. proposal is concerned with four fundamental issues: (i) the introduction of independence and locality in temporal specifications, so as to facilitate the study of properties of concurrent systems, (ii) the formulation of concurrent game procedures for satisfiability of fixpoint logics, (iii) the definition of concurrent game procedures for model checking such logics, and (iv) the development of game procedures to define equivalences between different models of concurrency, by means of studying the event structures obtained from the models used to give denotation to the fixpoint logics of interest in this dissertation.

2.2 Related Work

Independence in Logic. Most modal and temporal logics for concurrent systems, such as, LTL, CTL and the modal mu-calculi, have been defined using interleaving models of concurrency, where parallelism is modelled as the non-deterministic combination of all possible sequential executions of a concurrent computation. The information about actually local and independent concurrent computations of several processes is abstracted away and analysed in a global setting. As a consequence, there is no need in these logics to explicitly specify properties that must hold only in specific parts of a system (i.e., locally).

However, there exist logics that have been defined in order to capture aspects of independence and locality. For instance, IFML [3], ATL [2] and Separation Logic [25] are examples of such logical formalisms. Roughly speaking, these logics allow for the definition of formulae whose truth values depend on (or are relative to) only part of their underlying mathematical models. In the case of IFML and ATL this part of the model refers to information and capabilities that a specific player of a game has, whereas in Separation Logic the independence is reflected in that different formulae may be true in disjoint parts of its memory model. The notion of separation to model both locality and concurrency is the one of our interest.

This idea of defining separation properties to do local reasoning has also been investigated elsewhere and successfully applied in other settings. Separation as disjointness of resources was introduced to Computer Science in 1972 by Burstall in order to reason independently about programs [6]. Then, this idea was recasted by Reynolds et al. in [21, 25] to solve the long-standing problem of verifying the correctness of imperative programs with pointers, and by Pym and O'hearn to define a logical framework where classical and intuitionistic proofs could be combined [20].

Since then, due to the success of this approach to local reasoning, a great deal of work has been done in different areas by applying the notion of separation to reason independently about computer programs. For instance, in the specification of safety properties of concurrent programs [5, 11], the verification of while programs without concurrency [4, 27], the manipulation of tree structures [7], to reason about proofs as resources [23], or to model resource-sensitive synchronous processes [24], among others.

In [19], some of these works are discussed and several open problems are presented. One of them, which had not been addressed until now, was the problem of using this kind of local reasoning to specify and verify both linear-time and branching-time temporal properties. We here do so. The task is far from easy and require a heavy use of a mixture of concepts of true concurrency and fixpoint theory, besides the introduction of a new concept of locality, namely that of *support sets*. Locality and concurrency are therefore connected in our framework in terms of specification.

Concurrent and asynchronous games. Together with the problems in specification come those in verification. Tableau, Games and Automata are well-known and widely accepted approaches to the verification of concurrent systems. In [14, 15, 16, 28] games for satisfiability and model checking, among others, have been studied for various classes of

modal logics, fixpoint logics and temporal logics. There, the arenas are Kripke structures and Labelled transition systems, and the players, instead of taking alternated turns, interact with one another synchronously by performing a move at a time while checking the formulae at hand. This may not be the case when playing over true concurrency structures or partial orders and having operators for expressing independence and locality in the logic. Our hypothesis is that an asynchronous or concurrent version of the games investigated in [14, 15, 16, 28] will be required.

On the other hand, it can be found in the literature a number of asynchronous and concurrent games, but most of them are used in the context of game semantics and game graphs. In the former case, notably examples are the work by Abramsky and Mellies [1, 17]. Although these games are not design for verification, they are of our interest since they present an interesting approach to the formalisation of the strategies of the players involved in such games. Partial strategies based on *closure operators* on complete lattices is the key idea that allows the players to interact concurrently, in a synchronous and an asynchronous fashion.

Another approach to designing concurrent games is the one followed by Henzinger et al. [2, 8, 9]. These games are usually played by two agents who interact in a game graph by choosing independently and simultaneously moves that determine the future state of the game. This framework is particularly useful to model and verify reactive systems regarded as open ones. Unlike the games defined by Abramsky and Mellies, the agents who play in these games have strategies that are neither partial nor based on closure operators. Therefore, while in Abramsky's and Mellies's concurrent and asynchronous games the independence is in the form of the strategies, in Henzinger's concurrent games the independence is in the way that the decisions the players make are combined.

In this thesis both ways of defining concurrent games will be investigated, as each of these two approaches may suit better different problems. It is important to take into account that the three games to be studied here are played in completely different arenas, carrying different characteristics. *Games for satisfiability and completeness* are played only in fixpoint formulae, or rather the powerset of subformulae of a particular specification. They should be, therefore, valid for a fixpoint logic regardless its particular semantic model. On the other hand, *Games for model checking* are played in a formula and a mathematical structure (the model). Since the formulation of these games depends on the model at hand, there may be more restrictions on the kinds of strategies that can be defined. Finally, *Games for equivalence* are played in two different mathematical structures, the models of the behaviour (structures of events) of two concurrent systems. Our initial approach is to consider only a partially ordered representation of the events of such models. The arenas where these games are played are therefore two partial orders.

It is important to remark that the common approach to studying (behavioural) equivalences using games is that of defining procedures to check bisimilarities between two computational models. Several game characterisations of these procedures can be found in the literature. In concurrency theory and process algebras, the work presented in [28] by Stirling gives a good reference. The games for equivalence checking considered in [28] are essentially sequential and deal with Labelled transition systems. At this point, it is worth saying that a Labelled transition system is the interleaving model of concurrency closest to a Transitions systems with independence, one of the models to be used in this thesis.

Finally, other kinds of comparisons of several models of true concurrency have been already investigated in [22, 26, 29], but they follow categorical and automata-theoretic approaches, not a game-theoretical one as it is done in [28] for a model of interleaving concurrency, namely, for Labelled transition systems. In this thesis, we will focus our attention on *structural* models of concurrency [18], such as, Labelled transition systems with independence or Petri nets, for the purpose of modelling, so they will be used in the Games for Model checking. In contradistinction, *behavioural* models of concurrency

[18] based on partial orders will be used in the Games for equivalence checking. Our approach is therefore opposite (or perhaps complementary) to the one followed in [28], since a Labelled transition system is a structural model of concurrent computation, while a partial order of events is a behavioural one.

2.3 Dissertation outline

Chapter 1: Introduction. A brief introduction to the main topic and a motivation addressing the following questions: (1) What is the problem?; (2) Why is the problem important?; (3) What has so far been done on the problem?; (4) What is the contribution of the thesis on the problem?; (5) Why is the contribution original?; (6) Why is the contribution non-trivial?; and finally (7) a synopsis of the dissertation according to the content of the subsequent chapters.

Chapter 2: Background. In this chapter the main concepts and ideas about Modal and Temporal Logics, Games for Verification, and Concurrency Theory are introduced. This chapter will probably contain topics, such as, fixpoint modal logics and their models. Sequential games for Satisfiability and Model checking of such logics, as well as games for equivalence checking of some models of concurrency. Also, the main mathematical concepts to be used here will be introduced. For instance, lattices, partial orders and operators, relations and functions on these structures, such as, closure operators, Galois connections, and fixpoints.

Chapter 3: Separation Fixpoint Logic. In this chapter a fixpoint temporal logic (based on the modal mu-calculus) that is extended with ideas coming from Separation Logic is presented. We shall define its syntax and semantics, and highlight the most important features of the logic through examples. The main applications and characteristics of the logic will also be presented throughout the specification of truly concurrent systems.

Chapter 4: Games for Satisfiability. In this chapter we aim at defining games for satisfiability and completeness of fixpoint logics, similar to those defined in [15] for several other temporal logics. The games to be defined here are expected to be general enough to be used with other fixpoint logics, such as, the modal mu-calculi. Our initial approach is to define an asynchronous or concurrent version of the games presented in [15]. This chapter will present formal proofs of properties of the game procedure, such as, soundness, completeness, determinacy, winning strategies, complexity, and so forth. The study of these properties will actually take place for all the games defined in this dissertation.

Chapter 5: Games for Model checking. In this chapter we will define a class of games for model checking fixpoint logics with independence constructs and true concurrency structures as models. Similar to the previous chapter we will try to define an asynchronous or concurrent version of already known games for model checking temporal formulae. We aim to take advantage of the explicit separation or independence operators in the logic as well as their models with explicit concurrency, in order to avoid as much as possible the state explosion problem by doing local reasoning on true concurrency structures. The games defined here shall try to have in consideration the independence in both the logic and the models.

Chapter 6: Games for Equivalence checking. In this chapter we will define a class of games to study correspondences between different models of concurrency. These

games should be especially designed to identify similarities between partially ordered structures by identifying or using Galois connections between them. As in the previous chapters we will try to take advantage of the preorders where the games are played to define concurrent games on them. Also, similar to chapters 4 and 5, the main results of this chapter will be the proofs of the characteristics of the game above mentioned as well as an analysis of its usefulness.

Chapter 7: Conclusions and Future Work. This chapter will give a study or evaluation of the whole thesis by highlighting the advantages and disadvantages of exploiting the notion of independence for specifying and verifying temporal properties of concurrent systems. We may also elaborate on the philosophical implications of the results of this Ph.D. thesis and rise questions for future work. Furthermore, we will state goals that can be seen as (short/mid-term) further technical developments, which could not be accomplished due to time restrictions. They may be the starting point of a proposal for post-doctoral work.

2.4 Dissemination of results

The main results of this dissertation are expected to be sent for publication to some of the following international conferences: LICS, ICALP, CONCUR, CSL, MFCS, LPAR, STACS, VMCAI or FoSSaCS. A Journal publication containing the results achieved in the first two years will also be considered in the last year of the project.

3 Plan

3.1 First year

The first half of the academic year (from Sep/2007 to Mar/2008) was spent on background reading and attending some of the TPG and Master courses offered by LFCS and the School of Informatics. Besides this, I actively participated in the Complexity reading group organised by Dr. Mary Cryan, and gave a talk about *The Complexity of Petri nets* to the members of the group. I also gave a talk about *Games in the specification and verification of concurrent systems* to my former institution in Colombia, the Avispa research group. Moreover, I went to several courses offered by the Postgraduate Transferable Skills Unit of the University of Edinburgh, and attended the 24th British Colloquium for Theoretical Computer Science (BCTCS2008) in Durham, UK.

After this time, in the second half of the year (from Apr/2008 to Aug/2008), I have been focused on defining the proposal here presented, and working on its first specific objective. This objective is concerned with the definition of a fixpoint modal logic that captures aspects of independence and locality of concurrent systems. Our purpose is to be able to manipulate explicitly information about independence and locality of concurrent actions in parallel systems. This information is generally treated implicitly in other logics for concurrent computation, such as, LTL, CTL or the modal mu-calculus. Since our logic generalises temporal languages, such as, the modal mu-calculi, we expect some of our results (the games for the logic) to be valid for those other logics as well.

Although part of the results of this investigation is included in this document (the definition of the logic in Section 5), this is still work in progress. We expect to have, by Oct/2008, a paper to be sent for publication. Such a document should include the definition of the logic, the study of the equivalences induced by it as well as its main fragments (whose results are simply mentioned here), and, perhaps, the first approach to verifying properties in the logic - sequential games for model checking. Although we already started defining such games, due to space restrictions and the fact that we are in

the first stage of the definition of them, this work is not included in this report. All these results are to be sent to FoSSaCS at the beginning of October. A preliminary version of the results to be sent to this conference can be downloaded from [10].

3.2 Second year

The second year will be dedicated to work on the second and third specific objectives, namely, Games for satisfiability, completeness and model checking. We expect to have two research reports at the end of the second year (Dec/2009), one of them presenting an asynchronous or concurrent extension of the games for satisfiability and completeness developed by Stirling and Lange in [15], and the other report presenting asynchronous or concurrent games for model checking the fixpoint logics of our interest, i.e., the logic defined in Section 5 and the modal mu-calculus. These two documents will contain the definition of the games, results on their soundness and completeness, a study of their complexity, and some applications to both logics, so that it would be possible to compare our results when considering both an interleaving model of concurrency (for the mu-calculus), and a non-interleaving one (for our logic).

3.3 Third year

This year will be dedicated to work on the fourth specific objective (Games for equivalence checking) and writing up the final documents and reports of the dissertation. A separate research report will also be produced containing only the game procedures to be studied this year. Every research report produced every year is intended to be a separate chapter of the final document of the dissertation and a paper to be sent for publication to an international conference. This strategy of dissemination of results of the thesis aims at ensuring the quality of the work to be done in this Ph.D. project. It is easy to see that chapters 3 to 6 of the dissertation exactly match both a specific objective and a research report (which hopefully will become a paper as well).

4 Literature review

Since this thesis deal with topics related to Games, Logic and Concurrency in the context of the formal specification and verification of parallel systems, we shall give a brief, and sometimes informal, introduction to some Models of concurrency, Modal and Temporal Logics, and Games for solving their decision problems. However, we believe that more connexions between Logic, Games and Concurrency could be defined by studying the (sometimes implicit) notion of independence behind them. This connection may be identified through the existence of equilibrium points expressed as fixpoints in the logic, its models and the games to solve its decision procedures. For this reason, the second part of this literature review will introduce mathematical concepts that we believe will be relevant in the course of the thesis. This topics include partial orders, complete lattices, and monotone functions and relations on them, such as, closure operators and Galois connections. A brief summary of important fixpoint theorems in lattice and order theory will also be introduced.

4.1 Modal and Temporal Logics

Modal logics are mathematical formalisms that offer a new paradigm of applying logical methods: instead of using the traditional languages of quantification (first-order or higher-order) to describe a structure, they look for an appropriate quantifier-free language with additional logical operators (modalities) that represent the phenomenon at

hand. In a number of prominent cases, we end up with a logical language which is much richer than Boolean logic, and yet, unlike several languages with quantification, does not fall under the scope of classical undecidability limitations, thus often providing better decidability and complexity results than its rival first-order logic.

Also, Modal logics can be extended in very simple ways which may turn out to be extremely expressive. For instance, Modal logics can be used to express temporal properties by extending them with fixpoint operators. Notably, the modal mu-calculus is a small, yet expressive, temporal logic with modalities to distinguish and quantify actions or events performed by a system.

While Modal Logics are rather important in Mathematical Logic, *Temporal logics* play a major role in Informatics, especially in Concurrency theory. They are actually a special kind of modal logics. Temporal logics provide modalities for defining and reasoning about how the truth of an assertion changes over time. In general, temporal logics can be seen as logics with only a modality for a next step/time, and at least one operator to perform arbitrarily many sequences of steps. They can, therefore, be used to specify properties of the behaviour of a system in time.

Temporal logics come in two varieties: *linear time* and *branching time*. In a linear-time temporal logic, at each moment, there is only one possible future moment. On the other hand, in a branching-time temporal logic the possible future moments of time have a tree-like structure, so there may well be more than only one as in the linear-time case. The modalities of a temporal logic usually reflect the character of time assumed in the semantics of the logical language. Thus, in a linear-time temporal logic, the modalities are provided for describing events along a single time line. In contrast, in a branching-time temporal logic, the modalities reflect the branching nature of time by allowing quantification over various possible futures. It is difficult to say that one kind of temporal logic is better than the other. The usage of any of them depends on the sort of properties one would like to analyse.

4.2 Games for Modal and Temporal Logics

Several decision problems can be defined for modal and temporal logics. In the context of the temporal specification and verification of systems, two of them are of particular interest: the *Model checking* problem and the *Satisfiability* checking problem. Whereas the former asks if a given mathematical structure is a model of a given formula, the latter asks whether a given formula has a model that fulfills it.

In the last decade the use of Games to give a simple characterisation of these two problems has become more popular. In the games for model checking an *existential* and a *universal* player play on a formula and a graph structure, such as a Labelled transition system or a Kripke structure, in order to determine whether this graph structure fulfills the formula. On the other hand, in the games for satisfiability the same players interact with one another only on a formula so as to decide whether or nor the formula is satisfiable. Both problems reduce to the same question in a game-theoretic setting: does the existential player have a strategy to win every game?. Game-theoretic characterisations of these logical problems give rise to an interactive semantics of the modal and temporal logics studied. This is particularly useful in the temporal specification and verification of concurrent systems where games can be used to generate counterexamples to failing properties in a very natural way.

4.3 Models of Concurrency

In the theory of sequential computation, several mathematical formalisms have been proposed, e.g., Turing Machines, Lambda calculus, etc. A main result of this theory is

that all these formalisms are equivalent in the sense that their behaviour in terms of input-output functions is the same. For concurrent computation, in contradistinction, such an agreement does not exist yet, and various ways of defining the equivalence between different models of concurrent computation have been proposed. As a consequence, the question as to which the canonical model for concurrent computation is, remains still open. However, the models proposed so far can be distinguished depending on whether they model the *structure* or the *behaviour* of a system, and whether they model concurrency as a non-deterministic interleaving of actions or the parallel execution of events.

According to the features above mentioned, the most studied formalisms are Transition systems, Petri nets, Event structures and Process algebras. The two first computational models are graphical and may use labelling functions on their elements (e.g., nodes, arcs, places or transitions) to represent the systems they model. Event structures and Process Algebras, such as, CCS or CSP, on the other hand, are models of concurrency that abstract away from the actual structure of a system and focus on representing their behaviour. Although Transition systems and Process Algebras are usually regarded as models of interleaving concurrency, and Petri nets and Event structures as models of non-interleaving concurrency, most of them can be modified or adapted in order to capture either kind of behaviour.

4.4 Games for Models of Concurrency

In 1930 Alfred Tarski formulated the notion of two structures A and B being *elementarily equivalent*, i.e., that exactly the same first-order sentences are true in A as are true in B. Some years later, this idea was recasted and developed by logicians Roland Fraisse and Andrzej Ehrenfeucht in what is now known as the *Ehrenfeucht-Fraisse Games* or *Back-and-Forth* games. This games are used to compare mathematical and logical structures and have had a great impact in several areas in Mathematical Logic and Computer Science. Notably, the concept of bisimilarity, widely used in Concurrency to compare the behaviour of different parallel processes, is a class of Ehrenfeucht-Fraisse Games where the structures are said to be "elementary equivalent" in Tarski's words, if and only if the same modal formulae are true in both structures (i.e., in both models of concurrency).

In [28] these games, called *Equivalence checking Games*, or sometimes *Bisimilarity Games*, are presented and applied to compare parallel processes described using a process algebra. In an Equivalence checking game, an existential (Duplicator) and a universal (Spoiler) player play on two structures sequentially, one player after the other. As usual, the problem reduces to the question of whether the existential player has a winning strategy in every game, i.e., if Duplicator can always mimic in one of the two structures what Spoiler has just done in the other. The important point to make here is that the strategies that Duplicator and Spoiler use to play determine the equivalences between the structures where they play. In this way, modifications to their strategies can lead to the identification of different kinds of equivalences between the mathematical structures or models of concurrency being analysed.

4.5 Lattices and Complete lattices

Many important properties of an ordered set are expressed in terms of the existence of certain upper bounds or lower bounds of subsets of such a set. Two of the most important classes of ordered sets defined in this way are lattices and complete lattices.

A lattice P is a *partially ordered set* (poset) where for every two elements x and y in P, arbitrary meets or infimums ($x \wedge y$) and joins or supremums ($x \vee y$) exist. If, moreover, arbitrary meets and joins exist also for any subset $S \subseteq P$, then the lattice P is called a complete lattice.

This definition will be used in Section 5 when presenting the semantics of the fixpoint logic defined there. Also, lattices and complete lattices are relevant in this dissertation because most of the games to be defined in this thesis are played in these kinds of structures, either posets, lattices or complete lattices.

4.6 Closure operators and Galois connections

Another important topic related to partial orders refers to the functions and relationships that can be defined on their elements. These functions, some of which are presented in this section, can be used to characterise properties of the structures of our interest, and therefore, the arenas and strategies in the games for verification to be developed.

Closure Operators. Let X be a set. A closure operator on X is a map $C : \mathcal{P}(X) \to \mathcal{P}(X)$ from the powerset of X (a complete lattice) to itself, such that for all subsets A and B of S, the closure operator C satisfies the following three properties:

1. $A \subseteq C(A)$,
2. if $A \subseteq B$, then $C(A) \subseteq C(B)$,
3. $C(C(A)) = C(A)$

This means that a closure operator is a map that is (1) extensive, (2) monotonous and (3) idempotent. A fundamental property of a closure operator is that it is uniquely determined by its set of fixpoints, i.e., its set of closed elements $A = C(A)$.

It is also worth saying that every complete lattice arises (up to order-isomorphism) as the complete lattice of closed sets with respect to some closure operator.

Galois connections. Let $(X, \leq_X)$ and $(Y, \leq_Y)$ be two posets. A Galois connection (F, G) between the two posets is a pair of monotonic functions $F : X \to Y$ and $G : Y \to X$, such that for all $x \in X$ and $y \in Y$ we have that $x \leq_X G(y) \equiv F(x) \leq_Y y$.

Although, this definition is made on posets, the theory of Galois connections can be developed for preordered sets in general. It is also worth saying that part of the importance that Galois connections have had in many fields of Mathematics is due to the fact that, in many cases, a Galois connection allows one to express a relatively complex function (e.g., a strategy) in terms of another one, a relatively simpler one. This feature seems interesting when defining a game, as a Galois connection may help define well-behaved strategies for the players.

4.7 Fixpoint Theorems

Fixpoints can be seen as equilibrium points. Their mathematical definition is simple: $f(x) = x$. In mathematical logic they are particularly useful when the function f is monotone and the fixpoints are applied to members of a complete lattice with bottom element $\bot$ and top element $\top$, the smallest and largest elements of the lattice, respectively. As said before, a complete lattice is a non-empty, partially ordered set (poset) where arbitrary meets (infimum) and joins (supremum) exist. Finally, a point $a \in A$ is called a fixpoint of a function $f \in (A \to A)$ if $f(a) = a$. It is a pre-fixpoint of f if $f(a) \leq a$ and a post-fixpoint if $a \leq f(a)$. Suppose that $f : A \to A$ is a monotonic mapping on a complete lattice $(A, \leq)$. The following result holds:

Theorem 1. ***(Knaster-Tarski fixpoint theorem).*** If $f : A \to A$ is a monotonic mapping on a complete lattice $(A, \leq)$, then f has a least fixpoint $\text{fix}_\mu f$ and a greatest fixpoint $\text{fix}_\nu f$ which can be characterised as the smallest pre-fixpoint and largest post-fixpoint respectively:

$$\text{fix}_\mu f = \bigwedge\{a \in A \mid f(a) \leq a\} \text{ and } \text{fix}_\nu f = \bigvee\{a \in A \mid a \leq f(a)\}$$

If the function f is continuous there is a constructive way of characterising the least and greatest fixpoints by iterating the function on the bottom and top elements of the lattice, respectively. This can be stated, by using the two equations $f^0(a) = a$ and $f^{i+1}(a) = f(f^i(a))$, in the following way:

Theorem 2. ***(Kleene fixpoint theorem).*** If $f : A \to A$ is a continuous and monotonic mapping on a complete lattice $(A, \leq)$, then f has a least fixpoint $\text{fix}_\mu f$ and a greatest fixpoint $\text{fix}_\nu f$ which can be characterised as:

$$\text{fix}_\mu f = \bigvee\{f^i(\bot) \in A \mid i \geq 0\} \text{ and } \text{fix}_\nu f = \bigwedge\{f^i(\top) \in A \mid i \geq 0\}$$

So, we have that $f^0(\bot) \leq f^1(\bot) \leq f^2(\bot) \leq ...$ and, $f^0(\top) \geq f^1(\top) \geq f^2(\top) \geq ....$ This gives us an algorithm to calculate least and greatest fixpoints for continuous and monotonic functions on finite complete lattices: the least and greatest fixpoints of f are found exactly in the smallest i such that $f^i(\bot) = f^{i+1}(\bot)$ and $f^i(\top) = f^{i+1}(\top)$, respectively.

5 Separation Fixpoint Logic

We introduce Separation Fixpoint Logic (SFL for short), a modal logic that allows the specification of both *independence* and *locality* in temporal specifications, facilitating the identification of concurrent interactions between parallel processes. The logic, which subsumes many other linear-time and branching-time logics for concurrent computation, is a *substructural* language that treats concurrent actions as resources that can be locally recognisable.

The idea of independence in the logic is formalised with the introduction of a novel notion of locality called *support sets*. The logic makes use of operators that are sensitive to information in the support sets to naturally capture locally concurrent features of parallel processes. We give a structural, branching-time, non-interleaving semantic model for the denotation of Separation Fixpoint formulae, and show their usefulness when analysing properties of concurrent systems, especially, asynchronous ones since their natural models are true concurrency structures.

The main operators of this modal logic are a *parallel conjunction* and modal operators sensitive to locally independent actions. These operators, and their duals, are used to express actions performed by components different from those of interest as well as to separate the model in *disjoint* independent parts, so as to express that a property holds independently of others. These operators are further enriched with fixpoint constructions to express temporal properties by doing only local reasoning on independent actions.

In order to identify "responsible subsets" of transitions or events of a process action, we use the support sets. We say that a responsible transition can be regarded as possible at a given state whenever it is contained in the current support set. Otherwise the transition is regarded as non-supported. Whether a formula does hold or not is defined relatively to a state and a support set.

5.1 Syntax

Definition 5.1. Separation Fixpoint Logic (SFL) has formulae ϕ built from a set Var of variables $\mathrm{X}, \mathrm{Y}, \mathrm{Z}, ...$ and a set $\mathcal{L}$ of labels $a, b, ...$ by the following grammar:

$$\phi ::= Z \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \langle K\rangle_c\phi_1 \mid \langle K\rangle_{nc}\phi_1 \mid \phi_1 * \phi_2 \mid \mu Z.\phi_1$$

where Z is a variable, K ranges over subsets of $\mathcal{L}$, and $\mu Z.\phi_1$ has the restriction that any free occurrence of Z in ϕ_1 must be within the scope of an even number of negations.

Informally, the meaning of these operators is the following. "$\wedge$" and "$\neg$" are just booleans, "$\langle K\rangle_c$" (resp. "$\langle K\rangle_{nc}$") asserts at the fact that there is a causal (resp. a non-causal) action in the set of actions K that can be performed; $\phi_1 * \phi_2$ specifies that there exists a partition in the support set (i.e., a partition of the actions in the set to be considered) with which both formulae ϕ_1 and ϕ_2 can hold in parallel. This does not necessarily mean that both formulae hold in parallel everywhere because the operator "$*$" has a local meaning. However, with the use of fixpoint constructions truly parallel properties can be defined. Finally, "μ" is simply a least fixpoint operator.

Notation 5.2. The following abbreviations will be used throughout the document:

$$\begin{array}{lcl}
\mathrm{ff} & \stackrel{\text{def}}{=} & \mu Z.Z \\
\mathrm{tt} & \stackrel{\text{def}}{=} & \neg\mathrm{ff} \\
\phi_1 \vee \phi_2 & \stackrel{\text{def}}{=} & \neg(\neg\phi_1 \wedge \neg\phi_2) \\
\phi_1 \bowtie \phi_2 & \stackrel{\text{def}}{=} & \neg(\neg\phi_1 * \neg\phi_2) \\
[K]_c\,\phi_1 & \stackrel{\text{def}}{=} & \neg\langle K\rangle_c\neg\phi_1 \\
[K]_{nc}\,\phi_1 & \stackrel{\text{def}}{=} & \neg\langle K\rangle_{nc}\neg\phi_1 \\
\langle K\rangle\phi_1 & \stackrel{\text{def}}{=} & \langle K\rangle_c\phi_1 \vee \langle K\rangle_{nc}\phi_1 \\
[K]\,\phi_1 & \stackrel{\text{def}}{=} & [K]_c\,\phi_1 \wedge [K]_{nc}\,\phi_1 \\
\nu Z.\phi_1 & \stackrel{\text{def}}{=} & \neg\mu Z.\neg\phi_1\,[\neg Z/Z]
\end{array}$$

Another useful abbreviation will be $[-]\,\phi \stackrel{\text{def}}{=} [\mathcal{L}]\,\phi$ and $[-K]\,\phi \stackrel{\text{def}}{=} [\mathcal{L} - K]\,\phi$, and similarly for the other box ("[ ]") and diamond ("⟨ ⟩") operators.

Since positive normal form shall be assumed henceforth, explicit semantics for the dual boolean, modal, structural and fixpoint operators are given.

5.2 Semantics

The meaning of SFL formulae is given with respect to a Transition system with independence. A Transition Systems with Independence (TSI) [26] (TSI), is a simple structural, branching-time and noninterleaving extension of a Labelled Transition System (LTS), where independent events of a system can be explicitly recognised. A TSI is similar to a labelled *asynchronous* transition system (lats) [13], though while in the former the idea of event is derived from the independence between transitions, in the latter the events are basic elements of the structure. This very simple difference makes a TSI, in general, more adequate for giving Structural Operational Semantics to programming languages and process algebras. Nevertheless, both TSIs and lats are natural models of asynchronous processes, and therefore, parallel systems.

Transition Systems with Independence

Definition 5.3. *(Labelled Transition Systems (LTS)).* An LTS $\mathfrak{N}$ is a tuple (S, T, Σ) where S is a finite set of states, T is a transition relation such that $T \subseteq S \times \Sigma \times S$, where Σ is a finite set of action labels. We call this structure an LTS $\mathfrak{N}$ of sort Σ.

Definition 5.4. *(Transition Systems with Independence (TSI)).* A TSI $\mathfrak{T} = (\mathfrak{N}, I)$ is an LTS $\mathfrak{N} = (S, T, \Sigma)$ equipped an irreflexive and symmetric relation $I \subseteq T \times T$ such that:

- **A1**. $(s, a, s_1) \sim (s, a, s_2) \Rightarrow s_1 = s_2$
- **A2**. $(s, a, s_1) I (s, b, s_2) \Rightarrow \exists q.(s, a, s_1) I (s_1, b, q) \wedge (s, b, s_2) I (s_2, a, q)$
- **A3**. $(s, a, s_1) I (s_1, b, q) \Rightarrow \exists s_2.(s, a, s_1) I (s, b, s_2) \wedge (s, b, s_2) I (s_2, a, q)$
- **A4**. $(s, a, s_1) \prec \cup \succ (s_2, a, q) I (w, b, w') \Rightarrow (s, a, s_1) I (w, b, w')$

where $\prec$ is a binary relation on transitions such that:

$$(s, a, s_1) \prec (s_2, a, q) \Leftrightarrow \exists b.(s, a, s_1) I (s, b, s_2) \wedge (s, a, s_1) I (s_1, b, q) \wedge (s, b, s_2) I (s_2, a, q)$$

and $\sim$ is the least equivalence relation that includes $\prec$.

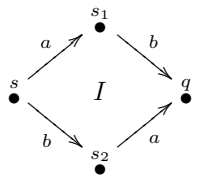


Figure 1: A concurrency diamond for $a\ I\ b$

The intuitions behind the axioms **A1**-**A4** as well as $\prec$ and $\sim$ follow from primitive ideas behind most noninterleaving models of concurrency. Let t_1 and t_2 be two transition in T. While $t_1 \prec t_2$ means that transitions t_1 and t_2 are part of the same "concurrency diamond", $t_1 \sim t_2$ means that transitions t_1 and t_2 are *occurrences* of the same event. Henceforth we will call occurrences of events as transitions and events as actions. Thus, transitions ought to be thought of as occurrences of actions. Regarding the axioms, **A1** states that from any state of the system, the execution of an action leads always to a unique future state. **A2** (resp. **A3**) says that if two actions can be executed in parallel (resp. one after the other), they can also be executed one after the other (resp. in parallel). Finally, **A4** ensures that the relation I between transitions is well defined. Roughly speaking **A4** says that if actions a and b are independent, then all the transitions that are occurrences of the action a are independent of all the transitions that are occurrences of the action b. Having said that, we can give an alternative, more intuitive, definition for axiom **A4**.

Let $\mathcal{I}(t)$ be the set $\{t' \mid tIt'\}$. Then, axiom **A4** is equivalent to the following expression:

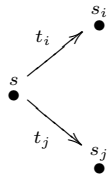
$$\textbf{A4}.\ t_1 \sim t_2 \Rightarrow \mathcal{I}(t_1) = \mathcal{I}(t_2)$$

Notation 5.5. Given a transition $t = (s_1, a, s_2)$, s_1 will be called the source node, $src(t) = s_1$; s_2 the target node, $trg(t) = s_2$; and a will be the label of such a transition, $lbl(t) = a$. A transition $t = (s_1, a, s_2)$ can also be written as $s_1 \xrightarrow{a} s_2$ or $s_1 \xrightarrow{t} s_2$.

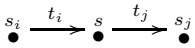
Concurrency, Causality and Conflict

The semantics of Separation Fixpoint Logic is based on the identification of two observations: in a local setting (immediate) *concurrency* is dual to *conflict*, and (linearized) *concurrency* is dual to *causality*. These two observations are defined formally in the

following way. Consider a state s of a concurrent system, say, a TSI, and any two different transitions t_i and t_j with the same source node, i.e., $src(t_i) = s = src(t_j)$. These two transitions are either concurrent, and therefore belong to I, or dependent, in which case they must be in conflict. Similarly, consider any two transitions t_i and t_j where $trg(t_i) = s = src(t_j)$. Again, these two transitions can either belong to I, in which case they are independent, yet have been linearized, or they do not belong to I, and therefore are causality dependent. These two dualities between concurrency and conflict, and concurrency and causality can be formalized for pairs of transitions as well as for sets of transitions in order to give the semantics of SFL formulae. It is easy to see that in Figure 2(a) transitions t_i and t_j are either concurrent or in conflict. Similarly, in Figure 2(b) transitions t_i and t_j are either concurrent or causally dependent. In both cases, the two conditions are exclusive and there are not other possibilities.



(a) Concurrency vs. Conflict



(b) Concurrency vs. Causality

Figure 2: Dualities in Concurrency

Definition 5.6. *(Linearized concurrency).* Consider any two transitions t_i and t_j where $trg(t_i) = s = src(t_i)$.. These two transitions are said to be concurrent but linearized iff $t_i I t_j$. We write $t_i \ominus_s t_j$ for such a relation.

Dually, causally dependent transitions can be defined.

Definition 5.7. *(Locally causal transitions).* Consider any two transitions t_i and t_j where $trg(t_i) = s = src(t_i)$.. These two transitions are said to be causally dependent iff $\neg(t_i I t_j)$. We write $t_i \leq_s t_j$ for such a relation.

This formalization of the duality between concurrency and conflict will be used to defined the semantics of the modal operators of SFL, $\langle K \rangle_c$ and $\langle K \rangle_{nc}$ (and their duals), later on. Note that this duality is defined only locally with respect to a state s of the underlying TSI. Now, we turn our attention to the duality between concurrency and conflict.

Definition 5.8. *(Locally disjoint sets and transitions).* Let t_i and t_j be any two transitions whose source node is s, i.e., $src(t_i) = s = src(t_j)$. We say that t_i is disjoint or independent of t_j, $t_i \boxtimes_s t_j$ iff $t_i I t_j$. This relation $\boxtimes_s$ on pairs of transitions can be defined for sets of transitions as well. Given a set of transitions E whose source node is s and where all transitions in it are independent of one another, a pair A and B of locally disjoint sets (independent or concurrent sets) can be defined from E as:

$$E = A \otimes B \text{ iff } A \neq \emptyset \wedge B \neq \emptyset \wedge \forall t_i, t_j \in (E \times E).t_i \neq t_j \Rightarrow t_i \boxtimes_s t_j$$

The dual of this relation can be defined straightforwardly.

Definition 5.9. *(Locally conflicting sets and transitions).* Let t_i and t_j be any two transitions whose source node is s, i.e., $src(t_i) = s = src(t_j)$. We say that t_i is locally in conflict with t_j with respect to state s, written as $t_i \#_s t_j$, if they are not locally disjoint, Formally, $t_i \#_s t_j$ iff $\neg(t_i \boxtimes_s t_j)$. This relation on transitions, which is dual to $\boxtimes_s$, can also be defined for sets of transitions as a dual of $\otimes$ in the following way:

$$E = A \oplus B \text{ iff } A = \emptyset \vee B = \emptyset \vee \exists t_i, t_j \in (E \times E).t_i \neq t_j \wedge t_i \#_s t_j$$

Separation and Support Sets for local reasoning.

Clearly, given any two set of transitions A and B relative to a state s, it is always the case that either $A \otimes B$ or $A \oplus B$. The definition of $\otimes$ and $\oplus$ is the basic ingredient of the semantics of the structural operators $*$ and $\bowtie$ of SFL. However, two more definitions must be first introduced. These definitions establish the way *support sets* are constructed and identified by using a *separation rule*.

Definition 5.10. *(Separation rule).* Given a non-empty set of transitions Q where $\forall t_i, t_j \in Q$ we have that $src(t_i) = src(t_j)$, a Separation rule finds a partition of the set Q iff there exists a non-empty set S such that $Q = S \oplus R$ and $S = M \otimes N$, where $S = M \cup N$ and $\emptyset = M \cap N$, and similarly, $Q = S \cup R$ and $\emptyset = S \cap R$. Therefore it is possible to write:

$$sep(Q) \Leftrightarrow \exists S \subseteq Q.S \neq \emptyset \wedge \exists R, M, N \subset Q.$$
$$Q = S \oplus R \wedge Q = S \cup R \wedge \emptyset = S \cap R \wedge S = M \otimes N \wedge S = M \cup N \wedge \emptyset = M \cap N$$

Remark. Intuitively, the sets M and N can be regarded as sets of actions that can *all* be performed in parallel while the set R is the set of transitions in conflict with either M or N. The separation rule identifies concurrent actions while throwing away conflicting ones. When the sets M and N can be obtained from a set of transitions Q, the following notation can be used: $M = sup_1(sep(Q))$ and $N = sup_2(sep(Q))$.

From these definitions the concept of support set can be formalized.

Definition 5.11. *(TSI Support sets).* Given a set s of a TSI, the *maximal* set $Q_{max}(s)$ of s is the set of all transitions t such that $src(t) = s$. A support set P of s is either its maximal set $Q_{max}(s)$ or any set $P_i = sup_i(sep(Q_{max}(s)))$ constructed using the separation rule. Given a TSI $\mathfrak{T}$ with set of states S, the set of all its support sets $\mathfrak{P}$ is defined as $\bigcup_{s \in S}\{P \in \mathfrak{P} \mid P = Q_{max}(s) \vee P = sup_1(sep(Q_{max}(s))) \vee P = sup_2(sep(Q_{max}(s)))\}$.

Denotation of SFL formulae

Definition 5.12. *(A TSI-based SFL model).* Given a TSI-based semantics, an SFL model $\mathfrak{M}$ is a Transition System with Independence $\mathfrak{T} = (S, T, \Sigma, I)$ together with a valuation $\mathcal{V} : \text{Var} \rightarrow 2^{\mathfrak{S}}$, where $\mathfrak{S} = S \times \mathfrak{P} \times T$ is the set of tuples (s, P, t_a) of states $s \in S$, support sets $P \in \mathfrak{P}$, and transitions $t_a \in T$, with action label $a \in \Sigma$ in the TSI $\mathfrak{T}$. The denotation $\|\phi\|^{\mathfrak{T}}_{\mathcal{V}}$ of an SFL formula ϕ in the model $\mathfrak{M} = (\mathfrak{T}, \mathcal{V})$ is given by the following rules (omitting the superscript $\mathfrak{T}$):

$$\begin{array}{lcl}
\|Z\|_{\mathcal{V}} & = & \mathcal{V}(Z) \\
\|\neg\phi\|_{\mathcal{V}} & = & \mathfrak{S} - \|\phi\|_{\mathcal{V}} \\
\|\phi_1 \wedge \phi_2\|_{\mathcal{V}} & = & \|\phi_1\|_{\mathcal{V}} \cap \|\phi_2\|_{\mathcal{V}} \\
\|\langle K\rangle_c \phi\|_{\mathcal{V}} & = & \{(s, P, t_a) \in \mathfrak{S} \mid \exists b \in K. \exists s' \in S. t_b = s \xrightarrow{b} s' \in P \wedge t_a \leq_s t_b \wedge (s', P', t_b) \in \|\phi\|_{\mathcal{V}}\} \\
\|\langle K\rangle_{nc} \phi\|_{\mathcal{V}} & = & \{(s, P, t_a) \in \mathfrak{S} \mid \exists b \in K. \exists s' \in S. t_b = s \xrightarrow{b} s' \in P \wedge t_a \ominus_s t_b \wedge (s', P', t_b) \in \|\phi\|_{\mathcal{V}}\} \\
\|\phi_1 * \phi_2\|_{\mathcal{V}} & = & \{(s, P, t_a) \in \mathfrak{S} \mid \exists P_1, P_2, P_3 \subset P. (P_1 \otimes P_2) \oplus P_3 \wedge (s, P_1, t_a) \in \|\phi_1\|_{\mathcal{V}} \wedge (s, P_2, t_a) \in \|\phi_2\|_{\mathcal{V}}\}
\end{array}$$

, where P' is the maximal set of s', $P_1 = sup_1(sep(P))$ and $P_2 = sup_2(sep(P))$.

Proposition 5.1. *The boolean, modal and structural operators of SFL, previously defined, are monotone in the complete lattice $\mathcal{P}(\mathfrak{S}) = 2^{\mathfrak{S}}$, the powerset of $\mathfrak{S}$, with respect to set inclusion $\subseteq$ and with bottom element $\perp = \emptyset$ and top element $\top = \mathfrak{S}$.*

Therefore, it is possible to define the denotation of the fixpoint operator $\mu Z.\phi(Z)$ according to the Knaster-Tarski fixpoint theorem:

$$\|\mu Z.\phi(Z)\|_{\mathcal{V}} \quad = \quad \bigcap\{Q \in \mathcal{P}(\mathfrak{S}) \mid \|\phi\|_{\mathcal{V}[Z:=Q]} \subseteq Q\}$$

where $\mathcal{V}[Z := Q]$ is the valuation $\mathcal{V}'$ which agrees with $\mathcal{V}$ save that $\mathcal{V}'(Z) = Q$.
We can now give the denotation of the dual operators:

$$
\begin{array}{lcl}
\|\phi_1 \vee \phi_2\|_{\mathcal{V}} & = & \|\phi_1\|_{\mathcal{V}} \cup \|\phi_2\|_{\mathcal{V}} \\
\|[K]_c\, \phi\|_{\mathcal{V}} & = & \{(s, P, t_a) \in \mathfrak{S} \mid \forall s' \in S. \forall b \in K. t_b = s \xrightarrow{b} s' \in P \wedge t_a \leq_s t_b \Rightarrow (s', P', t_b) \in \|\phi\|_{\mathcal{V}}\} \\
\|[K]_{nc}\, \phi\|_{\mathcal{V}} & = & \{(s, P, t_a) \in \mathfrak{S} \mid \forall s' \in S. \forall b \in K. t_b = s \xrightarrow{b} s' \in P \wedge t_a \ominus_s t_b \Rightarrow (s', P', t_b) \in \|\phi\|_{\mathcal{V}}\} \\
\|\phi_1 \bowtie \phi_2\|_{\mathcal{V}} & = & \{(s, P, t_a) \in \mathfrak{S} \mid \forall P_1, P_2, P_3 \subset P. (P_1 \otimes P_2) \oplus P_3 \Rightarrow ((s, P_1, t_a) \in \|\phi_1\|_{\mathcal{V}} \vee (s, P_2, t_a) \in \|\phi_2\|_{\mathcal{V}})\} \\
\|\nu Z. \phi(Z)\|_{\mathcal{V}} & = & \bigcup\{Q \in \mathcal{P}(\mathfrak{S}) \mid Q \subseteq \|\phi\|_{\mathcal{V}[Z:=Q]}\}
\end{array}
$$

, where P' is the maximal set of s', $P_1 = sup_1(sep(P))$ and $P_2 = sup_2(sep(P))$.

5.3 Applications and Syntactic SFL fragments

SFL is a powerful logic that allows one to express several kinds of temporal properties of parallel systems, such as, liveness, safety, fairness, and so on, due to the availability of *fixpoint operators*. Moreover, properties of true-concurrency structures, such as, causality and independence of parallel components, can also be expressed in the logic. This is due to a finer definition of *modal operators* and the introduction of a *parallel conjunction*.

It is also possible to combine all these operators so as to define rather complex properties, such as, the computation of causal lines, which require both fixpoint constructions and modal operators sensitive to causal information. They can further be studied by using the structural operators $*$ and $\bowtie$. Another application is the study of programs that share data, since the structural operators of SFL can be used to define separation properties in order to ensure the correct performance of such programs. Besides this, since the TSI model of SFL seems quite adequate to give a Structural Operational Semantics to resource-sensitive Process Algebras, such as the one presented in [24], SFL can be used to analyse temporal properties of such Process calculi taking into account an explicit notion of resource.

Logical and Concurrent Equivalences. However, one would like to restrict the expressive power of SFL for specific applications. We have identified three SFL sublogics which induce very natural equivalences in concurrency. These sublogics are syntactic fragments where the interplay between concurrency and conflict as well as concurrency and causality is restricted.

The first sublogic is obtained by disabling the sensitivity to both concurrency and causality, and concurrency and conflict. The logic turns out to have the same expressive power than the propositional modal mu-calculus, $L\mu$. Therefore, the equivalence induce by this fragment is exactly *Milner's strong bisimilarity*, $\sim_b$ [12]. The second fragment is the Causal Modal mu-calculus, $CL\mu$. This sublogic is obtained from $L\mu$ by allowing only the identification of the duality between concurrency and causality throughout the modal operators of the logic. The naturality of this logic for expressing causal properties is demonstrated by the equivalence it induces in any model, written as $\sim_{CL\mu}$, which coincides with a *History-preserving bisimilarity*, $\sim_{hpb}$, [30]. The third sublogic is the Separation Modal mu-calculus, $SL\mu$. This logic is obtained, again from $L\mu$, by allowing only the identification of the duality between concurrency and conflict throughout the structural operators of SFL. Although the equivalence induced by this logic, $\sim_{SL\mu}$, is in general incomparable with $\sim_{hpb}$, these two equivalences coincide for a large and important class of concurrent systems where concurrency and conflict are restricted. The systems we consider can be used as a model of race-free programs when dealing with processes that share information. The proofs of these results can be downloaded from [10].

Logical characterization of True-concurrency systems. Interestingly, although both the Separation mu-calculus and the Causal mu-calculus differentiate concurrency

from non-determinism, the each of them does it by using a different kind of duality. This is better explained with the following example.

Example 5.1. This very simple example, illustrated in Figure 3, shows that the only Separation and only Causal fragments of SFL can distinguish *concurrency* from *non-determinism*. The simplest way to see this is analysing the following two processes (in CCS-like notation): $P = a.0 \parallel b.0$ and $Q = a.b.0 + b.a.0$. They are of course different from a true-concurrency point of view. Such a difference can be captured in this two ways: with the formulae $\phi = \langle a\rangle \mathrm{tt} * \langle b\rangle \mathrm{tt}$ and $\psi = \langle a\rangle\langle b\rangle_{nc}\mathrm{tt}$, which are both true in the system on the left but false in the system on the right.

(a) $P = a.0 \parallel b.0$ (b) $Q = a.b.0 + b.a.0$

Figure 3: Concurrency vs. Non-determinism

The formulae presented in Example 5.1 show an important feature of SFL, that is, the fact that in a local setting SFL captures the two basic axioms of concurrency, which are illustrated by a concurrency diamond. These axioms are, first, that if two concurrent actions can be performed one after the other, then they can be executed in either order. This means that if formula $\psi = \langle a\rangle\langle b\rangle_{nc}\mathrm{tt}$ is true, then the formula $\psi' = \langle b\rangle\langle a\rangle_{nc}\mathrm{tt}$ must be also true. On the other hand, the second axiom of concurrency states that if two concurrent actions are both enabled at some state, then they both can be executed either order. This means that whenever $\phi = \langle a\rangle \mathrm{tt} * \langle b\rangle \mathrm{tt}$ is true, then the formulae $\psi = \langle a\rangle\langle b\rangle_{nc}\mathrm{tt}$ and $\psi' = \langle b\rangle\langle a\rangle_{nc}\mathrm{tt}$ must be true as well.

Although each SFL fragment seems to capture naturally some properties of concurrent systems, in some cases one would like to use the full expressive power of the logic to analyze some situations. For instance, the study and definition of causal lines. Certainly, they can be defined using just the Causal $L\mu$ fragment of SFL, but one would also like to study the independence relations between several causal lines. This can be done using the structural operators of SFL. In order to define causal lines, and other properties as well, derived operators for expressing strongly causal steps, which are preceded by non-causal ones, should be defined since strongly causal steps may require resource from a separate component in order to proceed.

$$\begin{array}{rcl} \langle K\rangle_{sc}\phi & \stackrel{\text{def}}{=} & \mu Z.\langle K\rangle_c\phi \vee \langle -\rangle_{nc} Z \\ [K]_{sc}\,\phi & \stackrel{\text{def}}{=} & \nu Z.\,[K]_c\,\phi \wedge [-]_{nc}\,Z \end{array}$$

Example 5.2. With the derived operators above defined, expressing that a property p eventually holds in a causal line from the initial state can be easily defined as $\phi = \mu Z.p \vee \langle -\rangle_{sc} Z$. The result of evaluating this formula corresponds to finding a "line" from the initial state to some other final state where p holds. This formula has, indeed, more information than the usual eventually operator $\Diamond$ of a standard temporal logic. These causal properties can be further study with the structural operators of SFL. Suppose the formula $\phi = \Diamond_{sc} p * \Diamond_{sc} q$, where $\Diamond_{sc} p$ stands for $\phi = \mu Z.p \vee \langle -\rangle_{sc} Z$, and similarly for $\Diamond_{sc} q$. This formula not only states that properties p and q hold eventually in two causal lines, but also that such causal lines have an independent source.

SFL as a resource-sensitive spatial logic. The parallel conjunction of SFL can also be used to define data structures, such as, lists and trees, *á la Reynolds*, but using a temporal specification approach. In Separation Logic, the states of a system can be treated as resources, so they can be separated, like in Girard's Linear Logic as well. Instead, in SFL independent transitions can be treated as resources, allowing indirectly the specification of independent localities in the model.

Example 5.3. Let $\phi = \mu Z.\langle emp\rangle \mathrm{tt} \vee (\langle data\rangle \mathrm{tt} * \langle - \rangle_{nc} Z)$. Formula ϕ represents a finite list structure where all data are spatially separated. Similarly, the specification of a tree can be stated as follows: $\psi = \mu Z.\langle emp\rangle \mathrm{tt} \vee (\langle data\rangle \mathrm{tt} * \langle - \rangle_{nc} Z * \langle - \rangle_{nc} Z)$.

These data structure specifications are similar to those presented in [4] with the difference that our approach is based on branching-time specifications while the approach in [4] is based on linear-time descriptions. Also, the notion of locality in [4] is the same as the one in Separation Logic, whereas ours relies on the identification of support sets of concurrent actions. In this way the combination of fixpoints and modal operators for expressing independence allows for simple descriptions of systems by doing local reasoning on concurrent transitions.

A logic for open systems?. By giving an adequate use to the various modalities and structural operators of SFL properties of *open systems* might also be stated. For instance, to express a property that holds inevitably for a component regardless the behaviour of any other parallel process. The main logic to express this sort of properties is ATL [2]. ATL is like CTL, but in the former the paths are explicitly manipulated. Its model is based on (Concurrent Synchronous) Games of perfect information. The main idea behind ATL is that the actions that are executed by an agent are analysed against what other concurrent components in the system can do. In our framework what a component of interest does (the system) may be represented as causal steps, whereas what other concurrent components do (the environment's behaviour) could be described as non-causal steps.

Example 5.4. Consider a specification for a process which we want to make sure always has the possibility to read and write from a shared memory. Such a specification can be expressed as $\phi = \Box(\Diamond_{sys} read \wedge \Diamond_{sys} write)$, where $\Box\psi_1 \stackrel{\text{def}}{=} \nu Y.\psi_1 \wedge ([-]Y \wedge \langle - \rangle \mathrm{tt})$ and $\Diamond_{sys}\psi_2 \stackrel{\text{def}}{=} \mu X.\psi_2 \vee \langle - \rangle_{sys} X$.

This example shows that the system can always read and write in the shared resource, no matter what the other processes in the system do. The operator $\langle a\rangle_{sys}\phi$ expresses an "adversarial possibility". Contrarily, the operator $\langle a\rangle_{sc}\phi$ (a strongly causal step) previously presented expresses a "collaborative possibility". The possibility of giving an operator like $\langle a\rangle_{sys}\phi$ with the intended semantics for open systems can be studied as well, since it seems feasible to be defined in our setting.

An advantage of using SFL to model open systems is in the specification of asynchronous systems. In order to properly model the behaviour of such systems, it is usually desirable to describe fairness properties of them, which are not possible to be expressed in some logics, like ATL, so the introduction of fairness constraints is required. In contradistinction, in SFL, as in the modal mu-calculus, fairness properties can be expressed, and therefore, there is no need to introduce any fairness constraints to the language.

5.4 Other logics for concurrency and local reasoning

In this report our purpose was to develop a logical language which captures explicitly the dualities between concurrency, causality and conflict in order to specify temporal

properties of parallel systems by introducing *independence and locality in temporal specifications*. The logic uses logical constructions for separation or independence, so as to specify locally properties of parallel sub-components of a system. In Separation Fixpoint Logic, concurrency and locality are captured by means of two different notions of local independence: support sets and modalities sensitive to causal information.

Although this is not the first time that local reasoning techniques similar to those in Separation Logic are used in the context of the logical specification and verification of properties [4, 5, 11, 24, 27], our work is the first one that can use it to describe and verify both linear-time and branching-time temporal properties of concurrent systems. This was possible due to the introduction of support sets as locally recognizable structures identifying concurrent interactions into a fixpoint modal language. The task is not easy since the combination of sub-structural operators and fixpoints is not trivial. This requires a deep understanding of fixpoint theory and monotonical reasoning, as well as the use of structural, branching-time and non-interleaving structures for concurrent computation.

The study and development of Separation Fixpoint Logic also rose several questions related to the specification and verification of asynchronous concurrent systems. For instance, the equivalences between true concurrent systems that can be captured by the logic, the question of whether local reasoning can be used to define asynchronous or concurrent games for verification, the possibility of giving a logical characterization to the progress (i.e., liveness properties) of resource-sensitive process algebras as the one presented in [24], or the possibility of defining a modal or temporal extension of Separation Logic itself by introducing the concept of support sets into it. Finally, an extremely intriguing problem is the question of whether the equivalences induced by Separation Fixpoint Logic could be used to give a characterization of different models of concurrency by using a logical approach instead of a categorical one, as done in [26].

References

[1] Samson Abramsky and Paul-André Melliès. Concurrent games and full completeness. In *LICS*, pages 431–442, 1999.

[2] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[3] Julian C. Bradfield and Sibylle B. Fröschle. Independence-friendly modal logic and true concurrency. *Nord. J. Comput.*, 9(1):102–117, 2002.

[4] Rémi Brochenin, Stéphane Demri, and Étienne Lozes. Reasoning about sequences of memory states. In Sergei N. Artëmov and Anil Nerode, editors, *LFCS*, volume 4514 of *Lecture Notes in Computer Science*, pages 100–114. Springer, 2007.

[5] Stephen D. Brookes. A semantics for concurrent separation logic. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 16–34. Springer, 2004.

[6] R. Burstall. Some techniques for proving correctness of programs which alter data structures. *Machine Intellingence*, 7:23–50, 1972.

[7] Cristiano Calcagno, Philippa Gardner, and Uri Zarfaty. Context logic and tree update. In Jens Palsberg and Martín Abadi, editors, *POPL*, pages 271–282. ACM, 2005.

[8] Luca de Alfaro and Thomas A. Henzinger. Concurrent omega-regular games. In *LICS*, pages 141–154, 2000.

[9] Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. *Theor. Comput. Sci.*, 386(3):188–217, 2007.

[10] Julian Gutierrez. `http://homepages.inf.ed.ac.uk/s0785782/separation.ps`. *Unpublished*. LFCS, School of Informatics, University of Edinburgh, 2008.

[11] Jonathan Hayman and Glynn Winskel. Independence and concurrent separation logic. In *LICS*, pages 147–156. IEEE Computer Society, 2006.

[12] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In J. W. de Bakker and Jan van Leeuwen, editors, *ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer, 1980.

[13] Thomas T. Hildebrandt and Vladimiro Sassone. Comparing transition systems with independence and asynchronous transition systems. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 1996.

[14] Martin Lange. *Games for Modal and Temporal Logics*. PhD thesis, University of Edinburgh, 2002.

[15] Martin Lange and Colin Stirling. Focus games for satisfiability and completeness of temporal logic. In *LICS*, pages 357–365, 2001.

[16] Martin Lange and Colin Stirling. Model checking games for branching time logics. *J. Log. Comput.*, 12(4):623–639, 2002.

[17] Paul-André Melliès. Asynchronous games 2: The true concurrency of innocence. *Theor. Comput. Sci.*, 358(2-3):200–228, 2006.

[18] Mogens Nielsen, Vladimiro Sassone, and Glynn Winskel. Relationships between models of concurrency. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX School/Symposium*, volume 803 of *Lecture Notes in Computer Science*, pages 425–476. Springer, 1993.

[19] Peter W. O'Hearn. Resources, concurrency, and local reasoning. *Theor. Comput. Sci.*, 375(1-3):271–307, 2007.

[20] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[21] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In Laurent Fribourg, editor, *CSL*, volume 2142 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2001.

[22] Vaughan R. Pratt. Transition and cancellation in concurrency and branching time. *Mathematical Structures in Computer Science*, 13(4):485–529, 2003.

[23] David J. Pym, Peter W. O'Hearn, and Hongseok Yang. Possible worlds and resources: the semantics of bi. *Theor. Comput. Sci.*, 315(1):257–305, 2004.

[24] David J. Pym and Chris M. N. Tofts. A calculus and logic of resources and processes. *Formal Asp. Comput.*, 18(4):495–517, 2006.

[25] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE Computer Society, 2002.

[26] Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Models for concurrency: Towards a classification. *Theor. Comput. Sci.*, 170(1-2):297–348, 1996.

[27] Élodie-Jane Sims. Extending separation logic with fixpoints and postponed substitution. *Theor. Comput. Sci.*, 351(2):258–275, 2006.

[28] Colin Stirling. *Modal and Temporal Properties of Processes*. Springer, 2001.

[29] Rob J. van Glabbeek. On the expressiveness of higher dimensional automata. *Theor. Comput. Sci.*, 356(3):265–290, 2006.

[30] Rob J. van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Inf.*, 37(4/5):229–327, 2001.