Model-Checking Games for Fixpoint Logics with Partial Order Models

Julian Gutierrez, Julian Bradfield

LFCS, School of Informatics, University of Edinburgh
Informatics Forum, 10 Crichton Street, Edinburgh, EH8 9AB, UK

Abstract

In this paper we introduce model-checking games that allow *local* second-order power on *sets* of independent transitions in the underlying partial order models where the games are played. Since the interleaving semantics of such models is not considered, some problems that may arise when using interleaving representations are avoided and new decidability results for partial order models of concurrency are achieved. The games are shown to be *sound* and *complete*, and therefore determined. While in the interleaving case they coincide with the local model-checking games for the μ-calculus, in a partial order setting they verify properties of a number of fixpoint modal logics that can specify, in concurrent systems with partial order semantics, several properties not expressible with the μ-calculus. The games underpin a novel decision procedure for model-checking all temporal properties of a class of infinite and regular event structures, thus improving, in terms of temporal expressive power, previous results in the literature.

Key words: Fixpoint modal logics, Model-checking games, Concurrency

1. Introduction

Model-checking games [12, 35], also called Hintikka evaluation games, are played by two players, a "Verifier" Eve ($\exists$) and a "Falsifier" Adam ($\forall$). These *logic games* [2] are played in a formula ϕ and a mathematical model $\mathfrak{M}$. In a game $\mathcal{G}(\mathfrak{M}, \phi)$ the goal of Eve is to show that $\mathfrak{M} \models \phi$, while the goal of Adam is to refute such an assertion. Solving these games amounts to answering the question of whether or not Eve has a strategy to win the game $\mathcal{G}(\mathfrak{M}, \phi)$. These games have a long history in mathematical logic and in the last two decades have become an active area of research in computer science, both from theoretical and practical view points. Good introductions to the subject can be found in [12, 33].

In concurrency and program verification, most usually ϕ is a modal or a temporal formula and $\mathfrak{M}$ is a Kripke structure or a labelled transition system (LTS), i.e., a graph structure, and the two players play the game $\mathcal{G}(\mathfrak{M}, \phi)$ *globally* by picking single elements of $\mathfrak{M}$, according to the game rules defined by ϕ. This setting works well for concurrent systems with *interleaving* semantics since one always has a notion of global state enforced by the nondeterministic sequential computation of atomic actions, which in turn allows the players to choose only single elements of

Email addresses: J.E.Gutierrez@ed.ac.uk (Julian Gutierrez), jcb@inf.ed.ac.uk (Julian Bradfield)

the structure $\mathfrak{M}$. However, when considering concurrent systems with partial order models [26], explicit notions of *locality* and *concurrency* have to be taken into account. A possible solution to this problem – the traditional approach – is to use the one-step interleaving semantics of such models in order to recover the *globality* and *sequentiality* of the semantics of formulae.

This solution is, however, problematic for at least five reasons. Firstly, interleaving models usually suffer from the state space explosion problem [4]. Secondly, interleaving interpretations cannot be used to give completely satisfactory game semantics to logics with partial order models as all information on independence in the models is lost in the interleaving simplification [1]. Thirdly, although temporal properties can still be verified with the interleaving simplification, properties involving concurrency, causality and conflict, natural to partial order models of concurrency, can no longer be verified [28]. From a more practical standpoint, partial order reduction methods [9, 11] or unfolding techniques [8] cannot be applied directly to interleaving models in order to build less complex model checkers based on these techniques. Finally, the usual techniques for verifying interleaving models cannot always be used to verify partial order ones since such problems may become undecidable [21, 27].

For these reasons, we believe that the study of verification techniques for partial order models continues to deserve much attention since they can help alleviate some of the limitations related with the use of interleaving models. We therefore abandon the traditional approach to defining model-checking games for logics with partial order models and propose a new class of games called 'trace local monadic second-order (LMSO) model-checking games', where *sets* of independent elements of the structure at hand can be *locally* recognized. These games avoid the need of using the one-step interleaving semantics of partial order models, and thus define a more natural framework for analysing fixpoint modal logics with noninterleaving semantics. Moreover, their use in the temporal verification of a class of regular event structures [34] improves previous results in the literature [21, 27]. We do so by allowing a free interplay of fixpoint operators and local second-order power on *conflict-free* sets of transitions.

The logic we consider is Separation Fixpoint Logic (SFL) [14], a μ-calculus (L_μ) [19] extension that can express causal properties in partial order models [26], e.g., transition systems with independence, Petri nets or event structures, and allows for doing *dynamic* local reasoning. The notion of locality in SFL, namely separation or disjointness of independent sets of resources, was inspired by the one defined *statically* for Separation Logic [29]. Since SFL is as expressive as L_μ in an interleaving context, nothing is lost with respect to the main approaches to logics for concurrency with interleaving semantics. Instead, logics and techniques for interleaving concurrency are extended to a partial order setting with SFL.

The structure of the paper is as follows: in Section 2 we introduce the partial order models of concurrency that are used in the paper and in Section 3 the syntax and semantics of SFL is defined. In Section 4, trace LMSO model-checking games are defined, and in Section 5 their soundness and completeness is proved. In Section 6, we show that the games are decidable and their coincidence with the local model-checking games for L_μ in the interleaving case. In Section 7 the game is used to effectively model-check a class of regular and infinite event structures. Finally, in Section 8 a summary of related work is given, and in Section 9 the paper concludes.

2. Preliminaries

This section introduces the background material that is needed in the following sections, namely the partial order models of our interest.

2.1. Partial Order Models of Concurrency

In concurrency there are two main approaches to modelling concurrent behaviour. On the one hand, interleaving models represent concurrency as the nondeterministic combination of all possible sequential behaviours in the system. On the other hand, partial order models represent concurrency explicitly by means of an independence relation on the set of actions, transitions or events in the system that can be executed concurrently.

We are interested in partial order models of concurrency for several reasons. In particular, because they can be seen as a generalisation of the interleaving models as will be explained later on in this section. This allows us to define the model-checking games presented here in a uniform way for several different models of concurrency, regardless of whether they have an interleaving or a partial order semantics. In the following, we present the three partial order models of concurrency that we consider here, namely Petri nets, transition systems with independence and event structures [26]. We also present some basic relationships between these three models, and how they generalize two important models for interleaving concurrency, which are also embraced in the uniform framework for model-checking we propose here. For further information the reader is referred to [26, 30] where one can find a more comprehensive presentation.

2.1.1. Petri Nets

A labelled *net* $\mathcal{N}$ is a tuple $(P, A, \mathcal{W}, \mathcal{F}, \Sigma)$, where P is a set of places, A is a set of actions, $\mathcal{W} \subseteq (P \times A) \cup (A \times P)$ is a relation between places and actions, and $\mathcal{F}$ is a labelling function $\mathcal{F} : A \to \Sigma$ from actions to a set Σ of action labels. Places and actions are called nodes; given a node n, ${}^\bullet n = \{x \mid (x, n) \in \mathcal{W}\}$ is the preset of n and $n^\bullet = \{y \mid (n, y) \in \mathcal{W}\}$ is the postset of n. These elements define the static structure of a net.[1] The notion of computation state in a net (its dynamic part) is that of a 'marking', which is a set or a multiset of places; in the former case such nets are called *safe*. Hereafter we only consider safe nets. Finally, a *Petri net* $\mathfrak{N}$ is a tuple $(\mathcal{N}, M_0)$, where $\mathcal{N} = (P, A, \mathcal{W}, \mathcal{F}, \Sigma)$ is a net and $M_0 \subseteq P$ is its initial marking.

As mentioned before markings define the dynamics of nets; they do so in the following way. We say that a marking M enables an action t iff ${}^\bullet t \subseteq M$. If t is enabled at M, then t can occur and its occurrence leads to a successor marking M', where $M' = (M \setminus {}^\bullet t) \cup t^\bullet$, written as $M \xrightarrow{t} M'$. Let $\xrightarrow{t}$ be the relation between all successive markings, and $\longrightarrow^*$ the reflexive and transitive closure of $\xrightarrow{t}$. Given a Petri net $\mathfrak{N} = (\mathcal{N}, M_0)$, the relation $\longrightarrow^*$ defines the set of *reachable* markings in the system $\mathfrak{N}$; such a set of reachable markings is fixed for any M_0 and can be constructed using the occurrence net unfolding of $\mathfrak{N}$ as defined in [25].

Finally, let par be the symmetric independence relation on actions such that $t_1 \ \mathsf{par} \ t_2$ iff ${}^\bullet t_1^\bullet \cap {}^\bullet t_2^\bullet = \emptyset$, where ${}^\bullet t^\bullet$ stands for the set ${}^\bullet t \cup t^\bullet$, and there exists a reachable marking M such that both ${}^\bullet t_1 \subseteq M$ and ${}^\bullet t_2 \subseteq M$. Then, if two actions t_1 and t_2 can occur concurrently they must be independent, i.e., $(t_1, t_2) \in \mathsf{par}$.

2.1.2. Transition Systems with Independence

A transition system with independence (TSI) is a labelled transition system (LTS) where independent transitions can be recognized. Formally, a TSI $\mathfrak{T}$ is a structure (S, s_0, T, I, Σ), where S is a set of states with initial state s_0, $T \subseteq S \times \Sigma \times S$ is a transition relation, Σ is a set of labels,

[1] The reader acquainted with net theory may have noticed that we use the word 'action' instead of 'transition', more common in the literature on (Petri) nets. We chose this notation in order to avoid confusion later on in the document.

and $I \subseteq T \times T$ is an irreflexive and symmetric relation on independent transitions. The binary relation $\prec$ on transitions defined by

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

expresses that two transitions are *instances* of the same a-labelled action, but in two different interleavings (see Figure 1 for a graphical description). We let $\sim$ be the least equivalence relation that includes $\prec$, i.e., the reflexive, symmetric and transitive closure of $\prec$. The equivalence relation $\sim$ is used to group all transitions that are instances of the same action in all its possible interleavings. Additionally, I is subject to the following axioms:

- **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**. $(s, a, s_1)\ (\prec \cup \succ)\ (s_2, a, q) \wedge (s_2, a, q)\ I\ (w, b, w') \Rightarrow (s, a, s_1)\ I\ (w, b, w')$

Axiom **A1** states that from any state, the execution of a transition leads always to a unique state. This is a determinacy condition. Axioms **A2** and **A3** ensure that independent transitions can be executed in either order. Finally, **A4** ensures that the relation I is well defined. More precisely, **A4** says that if two transitions t and t' are independent, then all other transitions in the equivalence class $[t]_\sim$ (i.e., all other transitions that are instances of the same action but in different interleavings) are independent of t' as well, and vice versa. Having said that, an alternative and possibly more intuitive definition for axiom **A4** can be given. Let $\mathcal{I}(t)$ be the set $\{t' \mid t\ I\ t'\}$. Then, axiom **A4** is equivalent to this expression: **A4'**. $t \sim t_2 \Rightarrow \mathcal{I}(t) = \mathcal{I}(t_2)$.

This axiomatization of concurrent behaviour was defined by Winskel and Nielsen [26], but has its roots in the theory of traces [22], notably developed by Mazurkiewicz for trace languages, one of the simplest partial order models of concurrency. As shown in Figure 1, this axiomatization can be used to generate a 'concurrency diamond' for any two independent transitions t and t', say, for $t = (s, a, s_1)$ and $t' = (s, b, s_2)$.

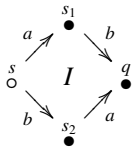

Figure 1: A concurrency diamond for $t\ I\ t'$. Concurrency or independence is recognized by the symbol I inside the square. The initial state of the TSI is marked by the circle $\circ$.

In a further (sub)section, we give a brief discussion about the relationships between this model and other mathematical formalisms, including languages and automata for concurrency.

Notation 2.1. *Given a transition $t = (s_1, a, s_2)$, also written as $s_1 \xrightarrow{a} s_2$ or $s_1 \xrightarrow{t} s_2$ if no confusion arises, the state s_1 is called the source node, the state $src(t) = s_1$; s_2 the target node, $trg(t) = s_2$; and a the label of t, $lbl(t) = a$.*

2.1.3. Event Structures

A labelled event structure $\mathfrak{E}$ is a tuple $(E, \preccurlyeq, \sharp, \eta, \Sigma)$, where E is a set of events that are partially ordered by $\preccurlyeq$, the causal dependency relation on events. Notice that events in an event structure are *occurrences* of actions in a system. Moreover $\sharp \subseteq E \times E$ is an irreflexive and symmetric conflict relation, and $\eta : E \to \Sigma$ is a labelling function such that the following hold:

$$\begin{array}{l} \text{If } e_1, e_2, e_3 \in E \text{ and } e_1 \sharp e_2 \preccurlyeq e_3, \text{ then } e_1 \sharp e_3. \\ \forall e \in E \text{ the set } \{e' \in E \mid e' \preccurlyeq e\} \text{ is finite.} \end{array}$$

The independence relation on events is defined with respect to the causal and conflict relations. Two events e_1 and e_2 are *concurrent*, denoted by $e_1 \ \mathsf{co} \ e_2$, iff $e_1 \not\preccurlyeq e_2$ and $e_2 \not\preccurlyeq e_1$ and $\neg(e_1 \sharp e_2)$. The notion of computation state for event structures is that of a *configuration*. A configuration C is a conflict-free set of events (i.e., if $e_1, e_2 \in C$, then $\neg(e_1 \sharp e_2)$) such that if $e \in C$ and $e' \preccurlyeq e$, then $e' \in C$. The initial configuration (or initial state) of any event structure $\mathfrak{E}$ is by definition the empty configuration $\{\}$. Finally, a successor configuration C' of a configuration C is given by $C' = C \cup \{e\}$ such that $e \notin C$. Write $C \xrightarrow{e} C'$ for this relation, and let $\longrightarrow^*$ be defined similarly to the Petri net case.

2.1.4. Towards a Unified View of Different Models of Concurrency

Despite being different informatic structures, the models of concurrency just presented have a number of fundamental relationships between them, as well as with some models for interleaving concurrency. More precisely, TSI are noninterleaving transition-based representations of Petri nets, whereas event structures are unfoldings of TSI. This is analogous to the fact that LTS are interleaving transition-based representations of Petri nets while trees are unfoldings of LTS.

On the other hand, there are also simple relationships between TSI and LTS as well as between event structures and trees in this way: LTS are exactly those TSI with an empty independence relation I on transitions, and trees are those event structures with an empty co relation on events. In this way, partial order models generalize the interleaving ones.

Since the results presented here are valid across all the models previously mentioned, it is convenient to fix some notations to refer unambiguously to any of them. To this end, we will use the notation coming from the TSI model and present the maps that determine a TSI model based on the primitives of the Petri net and event structure models. Also, with no further distinctions we use the word 'system' when referring to any of these models or to sub-models of them, e.g., an LTS or a Kripke structure.

The are two main reasons for this choice of notation. The first one is that the basic components of the TSI model can be easily and uniformly recognized in all the other models studied here. Thus, the translations are simple and direct. The second reason has to do with the fact that the concept of local dualities in partial order models, which is defined in the next section, can be presented explicitly in terms of the basic components of the TSI model.

Moreover, TSI models are preferred since they can be used to give a a noninterleaving semantics to CCS processes and related languages for concurrency [26]; in this way, simple CCS terms can be used to describe both finite and infinite TSI structures. TSI models also enjoy several interesting properties and are closely related to other models of concurrency, especially to 'asynchronous transition systems' [26] where TSI models appear as a well-structured subclass of systems. Through this relationship with asynchronous transition systems—which was studied in [17] using category theory tools—other connections can be found with many more models of concurrency [30] and even with automata theory, e.g., with Droste's concurrent automata [7], a kind of automata that generalizes asynchronous transition systems, and hence TSI models.

Just to recall, those components in the TSI model that can be identified uniformly in all other partial order models of concurrency are the following: a set S of states (with a uniquely defined initial state), a set T of labelled transitions between states, an independence relation I on elements of T, and an alphabet Σ of action labels.

TSI Representation of Petri Nets. A Petri net system $\mathfrak{N} = (\mathcal{N}, M_0)$, where $\mathcal{N} = (P, A, \mathcal{W}, \mathcal{F}, \Sigma)$ as defined before, can be represented as a TSI $\mathfrak{T} = (S, s_0, T, I, \Sigma)$ as follows:

$$\begin{array}{lcl} S & = & \{M \subseteq P \mid M_0 \longrightarrow^* M\} \\ T & = & \{(M, a, M') \in S \times \Sigma \times S \mid \exists t \in A.\ a = \mathcal{F}(t), M \xrightarrow{t} M'\} \\ I & = & \{((M_1, a, M_1'), (M_2, b, M_2')) \in T \times T \mid \exists (t_1, t_2) \in \mathsf{par}. \\ & & a = \mathcal{F}(t_1), b = \mathcal{F}(t_2), M_1 \xrightarrow{t_1} M_1', M_2 \xrightarrow{t_2} M_2'\} \end{array}$$

where S represents the set of reachable markings of the Petri net system $\mathfrak{N}$, the initial state s_0 is the initial marking M_0, and the set of labels Σ remains the same in both models.

TSI Representation of Event Structures. A TSI $\mathfrak{T} = (S, s_0, T, I, \Sigma)$ is determined by an event structure $\mathfrak{E} = (E, \preccurlyeq, \sharp, \eta, \Sigma)$ using the following mapping:

$$\begin{array}{lcl} S & = & \{C \subseteq E \mid \{\} \longrightarrow^* C\} \\ T & = & \{(C, a, C') \in S \times \Sigma \times S \mid \exists e \in E.\ a = \eta(e), C \xrightarrow{e} C'\} \\ I & = & \{((C_1, a, C_1'), (C_2, b, C_2')) \in T \times T \mid \exists (e_1, e_2) \in \mathsf{co}. \\ & & a = \eta(e_1), b = \eta(e_2), C_1 \xrightarrow{e_1} C_1', C_2 \xrightarrow{e_2} C_2'\} \end{array}$$

where S represents the set of configurations of the event structure $\mathfrak{E}$, the initial state s_0 is the initial configuration $\{\}$, and, as before, the set of labels Σ remains the same in both models. Notice that given this mapping from event structures to TSI, an infinite event structure would generate an infinite TSI. Since this is undesirable for model-checking purposes, in a later section we define a different mapping—from a class of infinite and regular event structures to TSI—that is better for model-checking as it always produces finite TSI representations.

Finally, also notice that *actions* in a Petri net, *transitions* in a TSI and *events* in an event structure are all different. As said before, transitions are *instances* of actions, i.e., are actions relative to a particular interleaving. On the other hand, events are *occurrences* of actions, i.e., are actions relative to the causality relation. However, they can all be analysed uniformly using a mathematical structure called a *process space*, which is to be defined in the following sections. Such a structure is used as a common bridge between different partial order models, and underlies the semantics of Separation Fixpoint Logic formulae, which we simply call 'SFL formulae'.

2.2. Local Dualities in Partial Order Models

We present two ways in which concurrency can be regarded as a dual concept to *conflict* and *causality*, respectively. These two ways of observing concurrency will be called *immediate concurrency* and *linearized concurrency*. Whereas immediate concurrency is dual to conflict, linearized concurrency is dual to causality. These local dualities were first defined in [14].

The intuitions behind these two observations are the following. Consider a concurrent system and any two different transitions t_1 and t_2 with the same source node, i.e., $src(t_1) = src(t_2)$. These two transitions are either immediately concurrent, and therefore independent, i.e., $(t_1, t_2) \in I$, or dependent, in which case they must be in conflict. Similarly, consider any two transitions t_1 and t_2 where $trg(t_1) = src(t_2)$. Again, the pair of transitions (t_1, t_2) can either belong to I, in which

case the two transitions are concurrent, yet have been linearized, or the pair does not belong to I, and therefore the two transitions are causally dependent. In both cases, the two conditions are exclusive and *there are no other possibilities*.

Notice that these dualities make sense only in a local setting. If two arbitrary transitions t_1 and t_2 do not have the property that $src(t_1) = src(t_2)$ or $trg(t_1) = src(t_2)$ (or vice versa), then nothing can be said about them doing only this analysis. However, as we will see later on, this simple notion of observation we introduce here is rather powerful since it is the basic ingredient for defining modal logics with partial order models.

The local dualities just described are formally defined in the following way, and notice the dual conditions between $\otimes$ and # and between $\ominus$ and $\leq$ with respect to the independence relation on transitions, if assuming valid the locality requirement:

$$\begin{aligned}
\otimes &\stackrel{\text{def}}{=} \{(t_1, t_2) \in T \times T \mid src(t_1) = src(t_2) \wedge t_1 \; I \; t_2\} \\
\# &\stackrel{\text{def}}{=} \{(t_1, t_2) \in T \times T \mid src(t_1) = src(t_2) \wedge \neg(t_1 \; I \; t_2)\} \\
\ominus &\stackrel{\text{def}}{=} \{(t_1, t_2) \in T \times T \mid trg(t_1) = src(t_2) \wedge t_1 \; I \; t_2\} \\
\leq &\stackrel{\text{def}}{=} \{(t_1, t_2) \in T \times T \mid trg(t_1) = src(t_2) \wedge \neg(t_1 \; I \; t_2)\}
\end{aligned}$$

Definition 2.2. (Local dualities) Let t_1 and t_2 be two transitions. We say that t_1 and t_2 are *immediately concurrent* iff $(t_1, t_2) \in \otimes$, *in conflict* iff $(t_1, t_2) \in \#$, *linearly concurrent* iff $(t_1, t_2) \in \ominus$, or *causally dependent* iff $(t_1, t_2) \in \leq$.

2.3. Sets in a Local Context

The relation $\otimes$ defined on pairs of transitions can be used to recognize *sets* where every transition is independent of each other and hence can all be executed concurrently. Such sets are said to be *conflict-free* and belong to the same *trace*, which is—following Mazurkiewicz trace theory—a conflict-free partially ordered set of actions, events or transitions for a given 'conflict' relation on the elements of such a set.

Definition 2.3. (Conflict-free sets) A *conflict-free set* of transitions P is a set of transitions with the same source node, where $t_1 \otimes t_2$ for any two distinct elements in P.

Notice that by definition empty and singleton sets are trivially conflict-free. Given a system $\mathfrak{T}$, all conflict-free sets of transitions at a state s can be defined locally from the *maximal set* of transitions $R_{\max}(s)$ consisting of all transitions t such that $src(t) = s$. We simply write $R_{\max}$ when the state s is defined elsewhere or is implicit from the context. Moreover, all maximal sets and conflict-free sets of transitions are fixed given a particular system $\mathfrak{T}$. Now we define the notion of locality used to give the semantics of the modal logics to be introduced in the next section.

Definition 2.4. (Support sets) Given a system $\mathfrak{T}$, a *support set* R in $\mathfrak{T}$ is either a maximal set of transitions in $\mathfrak{T}$ or a non-empty conflict-free set of transitions in $\mathfrak{T}$.

Given a system $\mathfrak{T}$, the set of all its support sets is denoted by $\mathfrak{P}$. As can be seen from the definition, support sets can be of two kinds, and one of them provides us with a way of doing local reasoning. More precisely, doing local reasoning on sets of independent transitions becomes possible when considering conflict-free sets since they can be *separated* or decomposed into smaller sets, where every transition is, as well, independent of each other.

Definition 2.5. (Complete traces) Given a support set R, a *complete trace* W of R, denoted by $W \sqsubseteq R$, is a support set $W \subseteq R$ such that $\neg\exists t \in R \setminus W.\ \forall t' \in W.\ t \otimes t'$.

It is easy to see that if R is a conflict-free support set, then W is R. However, if R is not a conflict-free support set, then R is necessarily a maximal set $R_{\max}$, and W must be a proper subset of R. Therefore, if $R = R_{\max}$, then the sets W such that $W \sqsubseteq R_{\max}$ are the biggest conflict-free support sets, which we call *maximal traces*, that can be recognized in a particular state s of a system $\mathfrak{T}$. Since all complete and maximal traces are support sets, then they are also fixed and computable given a system $\mathfrak{T}$.

3. Fixpoint Modal Logics

The local dualities and sets defined in the previous section can be used to build the semantics of a number of fixpoint modal logics which capture that behaviour of partial order models that is not present in interleaving one. As a consequence, these logics are more adequate languages for expressing properties of systems such as Petri nets, event structures or TSI.

The semantics of SFL is based on the recognition of what is actually *observable* in a partial order model. In other words, properties of system executions that are *conflict-free*. As defined by its semantics, SFL captures the duality between concurrency and causality by means of refining the usual modal operator of the μ-calculus, L_μ [19]. On the other hand, SFL captures the duality between concurrency and conflict with the use of a separating operator that behaves as a structural conjunction. This *structural operator* allows local reasoning on conflict-free support sets.

3.1. Process Spaces

Definition 3.1. (Process spaces) Let $\mathfrak{T} = (S, s_0, T, \Sigma, I)$ be a system, i.e., a partial order model as defined before. A *process space* $\mathfrak{S}$ is the lattice $S \times \mathfrak{P} \times \mathfrak{A}$, such that S is the set of states of $\mathfrak{T}$, $\mathfrak{P}$ is the set of support sets of $\mathfrak{T}$, and $\mathfrak{A}$ is the set of transitions $T \cup \{t_\epsilon\}$, where t_ϵ is the empty transition such that for all $t \in T$, $s_0 = src(t)$ iff $t_\epsilon \leq t$. A tuple $(s, R, t) \in \mathfrak{S}$ is called a *process*, and the initial process of $\mathfrak{S}$ is the tuple $(s_0, R_{\max}(s_0), t_\epsilon)$.

In practice one does not need to actually consider the whole lattice $S \times \mathfrak{P} \times \mathfrak{A}$, since support sets are defined with respect to a particular state. Therefore, if one knows the support set component of a process, then it is possible to infer the particular state in $\mathfrak{T}$.

3.2. Separation Fixpoint Logic

Definition 3.2. (SFL syntax) *Separation Fixpoint Logic* (SFL) has formulae ϕ built from a set Var of variables $Y, Z, ...$ and a set Σ of labels $a, b, ...$ by the following grammar:

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

where $Z \in \text{Var}$ 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. Dual boolean, modal, and fixpoint operators are defined in the usual way:

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

where $[\neg Z/Z]$ means substitution. Also, define the following derived operators: $\mathrm{ff} \stackrel{\text{def}}{=} \mu Z.Z$, $\mathrm{tt} \stackrel{\text{def}}{=} \neg\mathrm{ff}$, $\langle a\rangle\phi_1 \stackrel{\text{def}}{=} \langle a\rangle_c\phi_1 \vee \langle a\rangle_{nc}\phi_1$, $[a]\,\phi_1 \stackrel{\text{def}}{=} [a]_c\,\phi_1 \wedge [a]_{nc}\,\phi_1$. Using modal μ-calculus notation, the following abbreviations are also used: $\langle K\rangle$ for $\bigvee_{a\in K}\langle a\rangle$, where $K \subseteq \Sigma$, $[-]$ for $[\Sigma]$ and $[-K]$ for $[\Sigma \setminus K]$, and similarly for all other box and diamond modalities.

Informally, the meanings of the basic SFL operators are the following: $\wedge$ and $\neg$ are the usual boolean operators, $\langle a\rangle_c$ (resp. $\langle a\rangle_{nc}$) asserts that there is a causally dependent (resp. a non-causally dependent or linearly concurrent) transition with label a that can be performed; as defined in Section 2.2, such a transition is always either causally dependent or linearly concurrent w.r.t. the last transition that has been executed. $\phi_1 * \phi_2$ specifies that there exists a partition in the support set, i.e., a partition of the transitions in the set to be considered, w.r.t. which both formulae ϕ_1 and ϕ_2 can hold independently. This does not necessarily mean that both formulae hold in parallel everywhere because the operator $*$ has a local meaning. Finally, μ is simply a least fixpoint operator that allows the specification of recursive behaviour.

Definition 3.3. (SFL semantics) An *SFL model* $\mathfrak{M}$ is a system $\mathfrak{T} = (S, s_0, T, \Sigma, I)$ together with a valuation $\mathcal{V} : \mathrm{Var} \to 2^{\mathfrak{S}}$, where $\mathfrak{S} = S \times \mathfrak{P} \times \mathfrak{A}$ is the process space associated with $\mathfrak{T}$. The denotation $\|\phi_1\|^{\mathfrak{T}}_{\mathcal{V}}$ of an SFL formula ϕ in the model $\mathfrak{M} = (\mathfrak{T}, \mathcal{V})$ is a subset of $\mathfrak{S}$, given by the following rules (omitting the superscript $\mathfrak{T}$):

$$
\begin{array}{lcl}
\|Z\|_{\mathcal{V}} & = & \mathcal{V}(Z) \\
\|\neg\phi_1\|_{\mathcal{V}} & = & \mathfrak{S} - \|\phi_1\|_{\mathcal{V}} \\
\|\phi_1 \wedge \phi_2\|_{\mathcal{V}} & = & \|\phi_1\|_{\mathcal{V}} \cap \|\phi_2\|_{\mathcal{V}} \\
\|\langle a\rangle_c\phi_1\|_{\mathcal{V}} & = & \{(s,R,t) \in \mathfrak{S} \mid \exists s' \in S.\ \exists t' \in R. \\
 & & t' = s \xrightarrow{a} s' \wedge t \leq t' \wedge (s', R'_{\max}, t') \in \|\phi_1\|_{\mathcal{V}}\} \\
\|\langle a\rangle_{nc}\phi_1\|_{\mathcal{V}} & = & \{(s,R,t) \in \mathfrak{S} \mid \exists s' \in S.\ \exists t' \in R. \\
 & & t' = s \xrightarrow{a} s' \wedge t \ominus t' \wedge (s', R'_{\max}, t') \in \|\phi_1\|_{\mathcal{V}}\} \\
\|\phi_1 * \phi_2\|_{\mathcal{V}} & = & \{(s,R,t) \in \mathfrak{S} \mid \exists R_1, R_2 \in \mathfrak{P}. \\
 & & R_1 \uplus R_2 \sqsubseteq R \wedge (s, R_1, t) \in \|\phi_1\|_{\mathcal{V}} \wedge (s, R_2, t) \in \|\phi_2\|_{\mathcal{V}}\}
\end{array}
$$

where $R'_{\max}$ is the maximal set at s' and $\uplus$ means disjoint union of sets. Given the usual restriction on free occurrences of variables, imposed in order to obtain monotone operators in $\mathcal{P}(\mathfrak{S}) = 2^{\mathfrak{S}}$, the powerset lattice of $\mathfrak{S}$, it is possible to define the denotation of the fixpoint operator $\mu Z.\phi_1$ in the standard way, according to the Knaster-Tarski fixpoint theorem:

$$\|\mu Z.\phi_1\|_{\mathcal{V}} \quad = \quad \bigcap\{Q \subseteq \mathfrak{S} \mid \|\phi_1\|_{\mathcal{V}[Z:=Q]} \subseteq Q\}$$

where $\mathcal{V}[Z := Q]$ is the valuation $\mathcal{V}'$ which agrees with $\mathcal{V}$ save that $\mathcal{V}'(Z) = Q$. Since 'positive normal form' is assumed henceforth, the semantics of the dual boolean, modal, structural and fixpoint operators can be given in the usual way.

3.3. Examples

SFL can express all usual temporal properties, such as, liveness, safety, fairness, and so on, in systems with interleaving and partial order semantics.

Example 3.4. Let ϕ be the following reachability formula: $\phi = \mu Z.(\langle a\rangle_c\mathrm{tt} * \langle b\rangle_c\mathrm{tt}) \vee \langle -\rangle_c Z$. This SFL formula expresses the property that there exists an execution of causally dependent actions such that *eventually* two actions a and b can be executed in parallel. This specification is better than a similar one given by, e.g., the μ-calculus, since in the SFL case unnecessary interleavings are not checked and hence a combinatorial explosion of the state space to be searched is avoided.

On the other hand, since SFL is a logic for 'true-concurrency', it differentiates concurrency from nondeterminism in very simple ways. Consider the following systems (in CCS notation and with a partial order semantics, e.g., using TSI [26], Petri nets [6], or event structures [36]): $P = a \parallel b$ and $Q = a.b + b.a$. Processes P and Q, in Figure 2, are equivalent in an interleaving context (e.g., they are strongly bisimilar [16]), but different from a partial order viewpoint as they are not equated by any equivalence for true-concurrency. Such a difference can be captured, e.g., using the SFL formulae $\phi = \langle a \rangle \mathrm{tt} * \langle b \rangle \mathrm{tt}$ or $\psi = \langle a \rangle_c \langle b \rangle_{nc} \mathrm{tt}$, which are satisfied by P but not by Q.

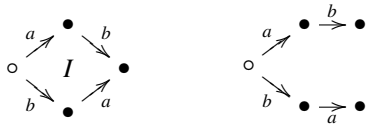

Figure 2: Interleaving vs. partial order (TSI) representations of $P = a \parallel b$ (on the left) and $Q = a.b + b.a$ (on the right).

Example 3.5. The strong distinguishing power of SFL allows the recognition of very subtle differences in the partial order behaviour of concurrent systems. For instance, SFL can differentiate some systems that are history-preserving (hp) bisimilar, but that are not hereditary history-preserving (hhp) bisimilar, such as those in Figure 3 (see [10] for a good reference on equivalences for true-concurrency).

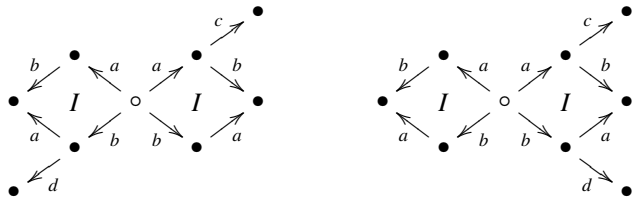

Figure 3: Concurrent systems with different partial order behaviour. Two hp bisimilar systems that are not hhp bisimilar.

4. Trace LMSO Model-Checking Games

Trace LMSO model-checking games $\mathcal{G}(\mathfrak{M}, \phi)$ are played on a model $\mathfrak{M} = (\mathfrak{T}, \mathcal{V})$, where $\mathfrak{T} = (S, s_0, T, I, \Sigma)$ is a system, and on an SFL formula ϕ. The game can also be presented as $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$, or even as $\mathcal{G}_{\mathfrak{M}}(s_0, \phi)$, where $H_0 = (s_0, R_{\max}(s_0), t_\epsilon)$ is the *initial process* of $\mathfrak{S}$. The board in which the game is played has the form $\mathfrak{B} = \mathfrak{S} \times \mathit{Sub}(\phi)$, where $\mathfrak{S}$ is the process space $S \times \mathfrak{P} \times \mathfrak{A}$ associated with $\mathfrak{T}$ and $\mathit{Sub}(\phi)$ is the subformula set of the SFL formula ϕ, which is defined by the Fischer–Ladner closure of SFL formulae in the standard way.

A play is a possibly infinite sequence of configurations $C_0, C_1, ...$, written as $(s, R, t) \vdash \phi$ or $H \vdash \phi$ whenever possible; each C_i is an element of the board $\mathfrak{B}$.[2] Every play starts in the

[2]Note that a configuration in a game is different from a configuration in an event structure. We use the same word and notation in both contexts for historical reasons. However, this is not a problem since confusion will never arise.

(FP) $\dfrac{H \vdash \sigma Z.\phi}{H \vdash Z}$	where $\sigma \in \{\mu, \nu\}$
(VAR) $\dfrac{H \vdash Z}{H \vdash \phi}$	for some $\sigma Z.\phi$
$(\vee)$ $\dfrac{H \vdash \phi_0 \vee \phi_1}{H \vdash \phi_i}$	$[\exists]\, i : \ \ i \in \{0,1\}$
$(\wedge)$ $\dfrac{H \vdash \phi_0 \wedge \phi_1}{H \vdash \phi_i}$	$[\forall]\, i : \ \ i \in \{0,1\}$
$(\langle\,\rangle_c)$ $\dfrac{(s,R,t) \vdash \langle a \rangle_c \phi}{(s', R'_{\max}(s'), t') \vdash \phi}$	$[\exists]\, a : \ \ t' = s \xrightarrow{a} s',\ t' \in R,\ t \leq t'$
$(\langle\,\rangle_{nc})$ $\dfrac{(s,R,t) \vdash \langle a \rangle_{nc} \phi}{(s', R'_{\max}(s'), t') \vdash \phi}$	$[\exists]\, a : \ \ t' = s \xrightarrow{a} s',\ t' \in R,\ t \ominus t'$
$([\,]_c)$ $\dfrac{(s,R,t) \vdash [a]_c\, \phi}{(s', R'_{\max}(s'), t') \vdash \phi}$	$[\forall]\, a : \ \ t' = s \xrightarrow{a} s',\ t' \in R,\ t \leq t'$
$([\,]_{nc})$ $\dfrac{(s,R,t) \vdash [a]_{nc}\, \phi}{(s', R'_{\max}(s'), t') \vdash \phi}$	$[\forall]\, a : \ \ t' = s \xrightarrow{a} s',\ t' \in R,\ t \ominus t'$
$(*)$ $\dfrac{(s,R,t) \vdash \phi_0 * \phi_1}{(s, R_i, t) \vdash \phi_i}$	$[\exists]\, R_0, R_1; [\forall]\, i : \ \ R_0 \uplus R_1 \sqsubseteq R, i \in \{0,1\}$
$(\bowtie)$ $\dfrac{(s,R,t) \vdash \phi_0 \bowtie \phi_1}{(s, R_i, t) \vdash \phi_i}$	$[\forall]\, R_0, R_1; [\exists]\, i : \ \ R_0 \uplus R_1 \sqsubseteq R, i \in \{0,1\}$

Figure 4: Trace LMSO Model-Checking Game Rules of SFL. Whereas the notation [∀] denotes a choice made by Player ∀, the notation [∃] denotes a choice by Player ∃.

configuration $C_0 = H_0 \vdash \phi$, and proceeds according to the rules of the game given in Figure 4. As usual for model-checking games, player ∃ tries to prove that $H_0 \models \phi$ whereas player ∀ tries to show that $H_0 \not\models \phi$.

The rules (FP) and (VAR) control the unfolding of fixpoint operators. Their correctness is based on the fact that $\sigma Z.\phi \equiv \phi\,[\sigma Z.\phi/Z]$ according to the semantics of the logic. Rules $(\vee)$ and $(\wedge)$ have the same meaning as the disjunction and conjunction rules, respectively, in a Hintikka game for propositional logic. Rules $(\langle\,\rangle_c)$, $(\langle\,\rangle_{nc})$, $([\,]_c)$ and $([\,]_{nc})$ are like the rules for quantifiers in a standard Hintikka game semantics for first-order (FO) logic, provided that the box and diamond operators behave, respectively, as restricted universal and existential quantifiers sensitive to the causal information in the partial order model.

Finally, the most interesting rules are $(*)$ and $(\bowtie)$. Local monadic second-order moves are used to recognize conflict-free sets of transitions in $\mathfrak{M}$, i.e., those in the same *trace*. Such moves, which restrict the second-order power (locally) to traces, give the name to this game. The use of $(*)$ and $(\bowtie)$ requires both players to make a choice: whereas the player who moves first must look for two conflict-free sets R_0 and R_1, the player that moves afterwards has to select a formula ϕ_i whose support set will be the corresponding R_i, for $i \in \{0,1\}$, just chosen by the other player.

Guided by the semantics of $*$ (resp. $\bowtie$), it is defined that player $\exists$ (resp. $\forall$) must look for a pair of non-empty conflict-free sets of transitions R_0 and R_1 to be assigned to each formula ϕ_i as their support sets. This situation is equivalent to playing a trace for each subformula in the configuration. Then player $\forall$ (resp. $\exists$) must choose one of the two subformulae, with full knowledge of the sets that have been given by player $\exists$ (resp. $\forall$). It is easy to see that $*$ should be regarded as a special kind of conjunction and $\bowtie$ of disjunction. Indeed, they are a structural conjunction and disjunction, respectively.

Definition 4.1. (Winning conditions) The following rules are the *winning conditions* that determine a unique winner for every finite or infinite play $C_0, C_1, ...$ in a game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$.

Player $\forall$ wins a finite play $C_0, C_1, ..., C_n$ or an infinite play $C_0, C_1, ...$ iff:

1. $C_n = H \vdash Z$ and $H \notin \mathcal{V}(Z)$.
2. $C_n = (s, R, t) \vdash \langle a \rangle_c \psi$ and $\{(s', R'_{\max}, t') : t \leq t' = s \xrightarrow{a} s' \in R\} = \emptyset$.
3. $C_n = (s, R, t) \vdash \langle a \rangle_{nc} \psi$ and $\{(s', R'_{\max}, t') : t \ominus t' = s \xrightarrow{a} s' \in R\} = \emptyset$.
4. $C_n = (s, R, t) \vdash \phi_0 * \phi_1$ and $\{(s, R_0 \cup R_1, t) : R_0 \uplus R_1 \sqsubseteq R\} = \emptyset$.
5. The play is infinite and there are infinitely many configurations where Z appears, such that Z is the least fixpoint of some subformula $\mu Z.\psi$ and the syntactically outermost variable in ϕ that occurs infinitely often.

Player $\exists$ wins a finite play $C_0, C_1, ..., C_n$ or an infinite play $C_0, C_1, ...$ iff:

1. $C_n = H \vdash Z$ and $H \in \mathcal{V}(Z)$.
2. $C_n = (s, R, t) \vdash [a]_c \psi$ and $\{(s', R'_{\max}, t') : t \leq t' = s \xrightarrow{a} s' \in R\} = \emptyset$.
3. $C_n = (s, R, t) \vdash [a]_{nc} \psi$ and $\{(s', R'_{\max}, t') : t \ominus t' = s \xrightarrow{a} s' \in R\} = \emptyset$.
4. $C_n = (s, R, t) \vdash \phi_0 \bowtie \phi_1$ and $\{(s, R_0 \cup R_1, t) : R_0 \uplus R_1 \sqsubseteq R\} = \emptyset$.
5. The play is infinite and there are infinitely many configurations where Z appears, such that Z is the greatest fixpoint of some subformula $\nu Z.\psi$ and the syntactically outermost variable in ϕ that occurs infinitely often.

In order to win a game, Player $\forall$ and Player $\exists$ make their choices according to their *strategies*. More precisely, a strategy for a player is a function which, given a play so far and a position where there is a choice, returns a specific choice and so tells the player how to move. A *winning* strategy is one which, if followed, guarantees that the player will win all plays of the game.

5. Soundness and Completeness.

Let us first give some intermediate results. The statements in this section are all either standard modal μ-calculus statements, or standard statements where additional cases for the new operators of SFL need to be checked. We give the statements in full, and the usual proof outlines, for the sake of being self-contained.

Let $\mathfrak{T}$ be a system and $C = (s, R, t) \vdash \psi$ a configuration in the game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$, as defined before. As usual, the denotation $\|\phi\|^{\mathfrak{T}}_{\mathcal{V}}$ of an SFL formula ϕ in the model $\mathfrak{M} = (\mathfrak{T}, \mathcal{V})$ is a subset of $\mathfrak{S}$. We say that a configuration C of $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$ is *true* iff $(s, R, t) \in \|\psi\|^{\mathfrak{T}}_{\mathcal{V}}$ and *false* otherwise.

Fact 5.1. *SFL is closed under negation.*

Lemma 5.2. *A game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$, where player $\exists$ has a winning strategy, has a dual game $\mathcal{G}_{\mathfrak{M}}(H_0, \neg\phi)$ where player $\forall$ has a winning strategy, and conversely.*

Proof. First, note that since SFL is closed under negation, for every rule that requires a player to make a choice on a formula ψ there is a dual rule in which the other player makes a choice on the negated formula $\neg\psi$. Also, note that for every winning condition for one of the players in a formula ψ there is a dual winning condition for the other player in $\neg\psi$. Now, suppose player $\exists$ has a winning strategy π in the game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$. Player $\forall$ can use π in the dual game $\mathcal{G}_{\mathfrak{M}}(H_0, \neg\phi)$ since whenever he has to make a choice, by duality, there is a rule that requires $\exists$ to make a choice in $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$. In this way, regardless of the choices that player $\exists$ makes, player $\forall$ can enforce a winning play for himself. The case when player $\forall$ has a winning strategy in the game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$ is dual. □

Lemma 5.3. *Player $\exists$ preserves falsity and can preserve truth with her choices. Hence, she cannot choose true configurations when playing in a false configuration. Dually, Player $\forall$ preserves truth and can preserve falsity with his choices. Then, he cannot choose false configurations when playing in a true configuration.*

Proof. The cases for the rules $(\wedge)$ and $(\vee)$ are just as for the Hintikka evaluation games for FO logic. Thus, let us go on to check the rules for the other operators. Firstly, consider the rule $(\langle\ \rangle_c)$ and a configuration $C = (s, R, t) \vdash \langle a\rangle_c\psi$, and suppose that C is false. In this case there is no a such that $t \leq t' = s \xrightarrow{a} s' \in R$, and $(s', R'_{\max}(s'), t') \in \|\psi\|^{\mathfrak{T}}_{\mathcal{V}}$. Hence, the following configurations will be false as well. Contrarily, if C is true, then player $\exists$ can make the next configuration $(s', R'_{\max}(s'), t') \vdash \psi$ true by choosing a transition $t' = s \xrightarrow{a} s' \in R$ such that $t \leq t'$. The case for $(\langle\ \rangle_{nc})$ is similar (simply change $\leq$ for $\ominus$), and the cases for $([\ ]_c)$ and $([\ ]_{nc})$ are dual. Now, consider the rule $(*)$ and a configuration $C = (s, R, t) \vdash \psi_0 * \psi_1$, and suppose that C is false. In this case there is no pair of sets R_0 and R_1 such that $R_0 \uplus R_1 \sqsubseteq R$ and both $(s, R_0, t) \in \|\psi_0\|^{\mathfrak{T}}_{\mathcal{V}}$ and $(s, R_1, t) \in \|\psi_1\|^{\mathfrak{T}}_{\mathcal{V}}$ to be chosen by player $\exists$. Hence, player $\forall$ can preserve falsity by choosing the $i \in \{0, 1\}$ where $(s, R_i, t) \notin \|\psi_i\|^{\mathfrak{T}}_{\mathcal{V}}$, and the next configuration $(s, R_i, t) \vdash \psi_i$ will be false as well. On the other hand, suppose that C is true. In this case, regardless of which i player $\forall$ chooses, player $\exists$ has previously fixed two support sets R_0 and R_1 such that for every $i \in \{0, 1\}$, $(s, R_i, t) \in \|\psi_i\|^{\mathfrak{T}}_{\mathcal{V}}$. Therefore, the next configuration $(s, R_i, t) \vdash \psi_i$ will be true as well. Finally, the deterministic rules (FP) and (VAR) preserve both truth and falsity because of the semantics of fixpoint operators. Recall that for any process H, if $H \in \|\sigma Z.\psi\|$ then $H \in \|\psi\|_{Z:=\|\sigma Z.\psi\|}$ for all free variables Z in ψ. □

Lemma 5.4. *In any infinite play of a game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$ there is a unique syntactically outermost variable that occurs infinitely often.*

Proof. By contradiction, assume that the statement is false. Without loss of generality, suppose that there are two variables Z and Y that are syntactically outermost and appear infinitely often. The only possibility for this to happen is that Z and Y are at the same level in ϕ. However, if this is the case Z and Y cannot occur infinitely often unless there is another variable X that also occurs infinitely often and whose unfolding contains both Z and Y. But this means that both Z and Y are syntactically beneath X, and therefore neither Z nor Y is outermost in ϕ, which is a contradiction. □

Fact 5.5. *Only rule* (VAR) *can increase the size of a formula in a configuration. All other rules decrease the size of formulae in configurations.*

Lemma 5.6. *Every play of a game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$ has a uniquely determined winner.*

Proof. Suppose the play is of finite length. Then, the winner is uniquely determined by one of the winning conditions one to four (Definition 4.1) of either player $\exists$ or player $\forall$ since such rules cover all possible cases and are mutually exclusive. Now, suppose that the play is of infinite length. Due to Fact 5.5, rule (VAR) must be used infinitely often in the game, and thus, there is at least one variable that is replaced by its defining fixpoint formula each time it occurs. Therefore, winning condition five of one of the players can be used to uniquely determine the winner of the game since, due to Lemma 5.4, there is a unique syntactically outermost variable that occurs infinitely often. □

Definition 5.7. (Approximants) *Let Z be the least fixpoint of some formula ϕ and let $\alpha, \lambda \in \mathbb{O}\mathrm{rd}$ be two ordinals, where λ is a limit ordinal. Then:*

$$Z^0 := \mathrm{ff}, \qquad Z^{\alpha+1} = \phi\,[Z^\alpha/Z], \qquad Z^\lambda = \bigvee_{\alpha<\lambda} Z^\alpha$$

For greatest fixpoints the approximants are defined dually. Let Z be the greatest fixpoint of some formula ϕ and, as before, let $\alpha, \lambda \in \mathbb{O}\mathrm{rd}$ be two ordinals, where λ is a limit ordinal. Then:

$$Z^0 := \mathrm{tt}, \qquad Z^{\alpha+1} = \phi\,[Z^\alpha/Z], \qquad Z^\lambda = \bigwedge_{\alpha<\lambda} Z^\alpha$$

We can now show that the analysis for fixpoint modal logics [3] can be extended to this scenario. The proof of soundness uses similar arguments to that in the μ-calculus case, but we present it here in full because it is the basis of the decision procedure for SFL model-checking.

Theorem 5.8. (Soundness) *Let $\mathfrak{M} = (\mathfrak{T}, \mathcal{V})$ be a model of a formula ϕ in the game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$. If $H_0 \notin \|\phi\|^{\mathfrak{T}}_{\mathcal{V}}$ then player $\forall$ wins $H_0 \vdash \phi$.*

Proof. Suppose $H_0 \notin \|\phi\|^{\mathfrak{T}}_{\mathcal{V}}$. We construct a possibly infinite game tree that starts in $H_0 \vdash \phi$, for player $\forall$. We do so by preserving falsity according to Lemma 5.3, i.e., whenever a rule requires player $\forall$ to make a choice then the tree will contain the successor configuration that preserves falsity. All other choices that are available for player $\exists$ are included in the game tree.

First, consider only finite plays. Since player $\exists$ only wins finite plays that end in true configurations, then she cannot win any finite play by using her winning conditions one to four. Hence, player $\forall$ wins each finite play in this game tree.

Now, consider infinite plays. The only chance for player $\exists$ to win is to use her winning condition five. So, let the configuration $H \vdash \nu Z.\phi$ be reached such that Z is the syntactically outermost variable that appears infinitely often in the play according to Lemma 5.4. In the next configuration $H \vdash Z$, variable Z is interpreted as the least approximant Z^α such that $H \notin \|Z^\alpha\|^{\mathfrak{T}}_{\mathcal{V}}$ and $H \in \|Z^{\alpha-1}\|^{\mathfrak{T}}_{\mathcal{V}}$, by the principle of fixpoint induction. As a matter of fact, by monotonicity and due to the definition of fixpoint approximants it must also be true that $H \in \|Z^\beta\|^{\mathfrak{T}}_{\mathcal{V}}$ for all ordinals β such that $\beta < \alpha$. Note that, also due to the definition of fixpoint approximants, α cannot be a limit ordinal λ because this would mean that $H \notin \|Z^\lambda = \bigwedge_{\beta<\lambda} Z^\beta\|^{\mathfrak{T}}_{\mathcal{V}}$ and $H \in \|Z^\beta\|^{\mathfrak{T}}_{\mathcal{V}}$ for all $\beta < \lambda$, which is impossible.

Since Z is the outermost variable that occurs infinitely often and the game rules follow the syntactic structure of formulae, the next time that a configuration $C' = H' \vdash Z$ is reached, Z can be interpreted as $Z^{\alpha-1}$ in order to make C' false as well. And again, if $\alpha - 1$ is a limit ordinal λ, there must be a $\gamma < \lambda$ such that $H' \notin \|Z^\gamma\|^{\mathfrak{T}}_{\mathcal{V}}$ and $H' \in \|Z^{\gamma-1}\|^{\mathfrak{T}}_{\mathcal{V}}$. One can repeat this process even until $\lambda = \omega$.

But, since ordinals are well-founded the play must eventually reach a false configuration $C'' = H'' \vdash Z$ where Z is interpreted as Z^0. And, according to Definition 5.7, $Z^0 := \mathrm{tt}$, which leads to a contradiction since the configuration $C'' = H'' \vdash \mathrm{tt}$ should be false, i.e., $H'' \in \|\mathrm{tt}\|^{\mathfrak{T}}_{\mathcal{V}}$ should be false, which is impossible. In other words, if H had failed a maximal fixpoint, then there must have been a descending chain of failures, but, as can be seen, there is not.

As a consequence, there is no such least α that makes the configuration $H \vdash Z^\alpha$ false, and hence, the configuration $H \vdash \nu Z.\phi$ could not have been false either. Therefore, player $\exists$ cannot win any infinite play with her winning condition 5 either. Since player $\exists$ can win neither finite plays nor infinite ones whenever $H_0 \notin \|\phi\|^{\mathfrak{T}}_{\mathcal{V}}$, then player $\forall$ must win all plays of $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$. □

Remark 5.9. If only finite state systems are considered $\mathbb{O}\mathrm{rd}$, the set of ordinals, can be replaced by $\mathbb{N}$, the set of natural numbers.

Notice that, in our setting, the previous remark is particularly important when the system $\mathfrak{T}$ in a model $\mathfrak{M}$ is the TSI representation of an event structure, since any concurrent system featuring recursive behaviour would be represented by an infinite event structure, and hence, by an infinite-state TSI model, if one uses the mapping from event structures to TSI given previously. Therefore, in this setting, we have to consider the possibility of dealing with infinite-state systems in order for the results of this section to apply to all the partial order models we presented in Section 2, as well as to the interleaving models they generalize.

Theorem 5.10. (Completeness) *Let $\mathfrak{M} = (\mathfrak{T}, \mathcal{V})$ be a model of a formula ϕ in the game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$. If $H_0 \in \|\phi\|^{\mathfrak{T}}_{\mathcal{V}}$ then player $\exists$ wins $H_0 \vdash \phi$.*

Proof. Suppose that $H_0 \in \|\phi\|^{\mathfrak{T}}_{\mathcal{V}}$. Due to Fact 5.1 it is also true that $H_0 \notin \|\neg\phi\|^{\mathfrak{T}}_{\mathcal{V}}$. According to Theorem 5.8, player $\forall$ wins $H_0 \vdash \neg\phi$, i.e., has a winning strategy in the game $\mathcal{G}_{\mathfrak{M}}(H_0, \neg\phi)$. And, due to Lemma 5.2, player $\exists$ has a winning strategy in the dual game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$. Therefore, player $\exists$ wins $H_0 \vdash \phi$ if $H_0 \in \|\phi\|^{\mathfrak{T}}_{\mathcal{V}}$. □

Theorems 5.8 and 5.10 imply that the game is determined. Determinacy and perfect information make the notion of truth defined by this Hintikka game semantics coincide with its Tarskian counterpart.

Corollary 5.11. (Determinacy) *Player $\forall$ wins the game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$ iff player $\exists$ does not win the game $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$.*

6. Local Properties and Decidability

We have shown that trace LMSO model-checking games are still sound and complete even when players are allowed to manipulate sets of independent transitions. Importantly, the power of these games, and also of SFL, is that such a second-order quantification is kept both *local* and restricted to transitions in the same *trace*. We now show that trace LMSO model-checking games enjoy several local properties that in turn make them *decidable* in the finite case. Such a decidability result is used in the forthcoming sections to extend the decidability border of model-checking a category of partial order models of concurrency.

Proposition 6.1. (Winning strategies) *The winning strategies for the trace LMSO model-checking games of Separation Fixpoint Logic are history-free.*

Proof. Consider a winning strategy π for player $\exists$. According to Lemma 5.3 and Theorem 5.10 such a strategy consists of preserving truth with her choices and annotating variables with their approximant indices. But neither of these two tasks depends on the history of a play. Instead they only depend on the current configuration of the game. In particular notice that, of course, this is also the case for the structural operators since the second-order quantification has only a local scope. Similar arguments apply for the winning strategies of player $\forall$. □

Remark 6.2. Corollary 5.11 and Proposition 6.1 also follow from the fact that the trace LMSO model-checking games for SFL are a form of parity games with perfect information.

This result is key to achieve *decidability* of these games in the presence of the local second-order quantification on the traces of the partial order models we consider. Also, from a more practical standpoint, memoryless strategies are desirable as they are easier to synthesize.

Theorem 6.3. *The model-checking game for finite systems against Separation Fixpoint Logic specifications is decidable.*

Proof. Recall that a game is decidable if one can tell in all possible cases which of the two players has a winning strategy in the game. Since the game is determined, finite plays are decided by winning conditions one to four of either player. Now consider the case of plays of infinite length; since the winning strategies of both players are history-free, we only need to look at the set of different configurations in the game, which is finite even for plays of infinite length. Now, in a finite system an infinite play can only be possible if the model is cyclic. But, since the model has a finite number of states, there is an upper bound on the number of fixpoint approximants that must be calculated (as well as on the number of configurations of the game board that must be checked) in order to ensure that either a greatest fixpoint is satisfied or a least fixpoint has failed. As a consequence, all possible history-free winning strategies for a play of infinite length can be computed, so that the game can be decided using winning condition five of one of the players. □

Remark 6.4. The complexity of model-checking is in principle substantially worse than for L_μ, but in practice not. The change in complexity from plain L_μ arises from the local second-order quantification in the $*$ operator – in principle, this could involve choosing a partition of a set of the order of the size of the state space, making the $*$ operation NP in the state space; hence the complexity for a formula of length k and alternation depth d on a system of size n is $O(kn.2^{nd})$ with the simple algorithms (or $O(kn.2^{nd/2})$ using the Browne et al. optimization). This maximal complexity occurs in highly concurrent systems, where it is the inevitable manifestation of state explosion. For typical systems encountered in reality, where the concurrency is small compared to the overall size, the support sets will be much smaller than the size of the system. Hence for practical purposes, the complexity is unlikely to be significantly worse than that of L_μ.

6.1. The Interleaving Case.

Local properties of trace LMSO model-checking games can also be found in the interleaving case, namely, they coincide with the local model-checking games for the modal μ-calculus as defined by Stirling [32]. Notice that interleaving systems can be cast using SFL by both syntactic and semantic means. The importance of this feature of SFL is that even having constructs for independence and a partial order model, nothing is lost with respect to the main approaches to interleaving concurrency. For instance, L_μ can be obtained from SFL by considering the

$*$-free language and using only the following derived operators: $\langle a\rangle\phi = \langle a\rangle_c\phi \vee \langle a\rangle_{nc}\phi$ and $[a]\,\phi = [a]_c\,\phi \wedge [a]_{nc}\,\phi$.

Proposition 6.5. *If either the class of models is restricted to those with an empty independence relation, or the class of formulae is restricted to L_μ, then the trace LMSO model-checking games for SFL degenerate to the local model-checking games for the μ-calculus.*

Proof. Let us consider the case when the syntactic L_μ fragment of SFL is considered. The first observation to be made is that the $*$-free fragment of SFL only considers *maximal sets*. Hence if a transition can be performed at s then it is always in the support set at s. Therefore, support sets in $\mathfrak{P}$ can be disregarded. Also, without loss of generality, consider only the case of the modal operators since the L_μ and SFL boolean and fixpoint operators have the same denotation.

$$\begin{aligned}\|\langle a\rangle\phi\|^{\mathfrak{T}}_{\mathcal{V}} \quad &= \quad \{(s,t) \in S \times \mathfrak{A} \mid \exists s' \in S.\ t \leq t' = s \xrightarrow{a} s' \wedge (s',t') \in \|\phi\|^{\mathfrak{T}}_{\mathcal{V}}\} \\ &\quad \cup \{(s,t) \in S \times \mathfrak{A} \mid \exists s' \in S.\ t \ominus t' = s \xrightarrow{a} s' \wedge (s',t') \in \|\phi\|^{\mathfrak{T}}_{\mathcal{V}}\}\end{aligned}$$

The second observation is that when computing the semantics of the combined operator $\langle a\rangle$, the conditions $t \leq t'$, i.e., $(t,t') \notin I$, and $t \ominus t'$, i.e., $(t,t') \in I$, complement each other and become always true (since there are no other possibilities). Therefore, the second component of every pair in $S \times \mathfrak{A}$ can also be disregarded.

$$\|\langle a\rangle\phi\|^{\mathfrak{T}}_{\mathcal{V}} \quad = \quad \{s \in S \mid \exists s' \in S.\ s \xrightarrow{a} s' \wedge s' \in \|\phi\|^{\mathfrak{T}}_{\mathcal{V}}\}$$

The case for the box operator $[a]$ is similar. Now, note that the new game rules and winning conditions enforced by these restrictions coincide with the ones defined by Stirling for the local model-checking games of L_μ. In particular, the new game rules and winning conditions for the modalities are as follows.

In a finite play $C_0, C_1, ..., C_n$ of $\mathcal{G}_{\mathfrak{M}}(H_0, \phi)$, where C_n has a modality as a formula component, player $\forall$ wins iff $C_n = s \vdash \langle a\rangle\psi$ and $\{s' : s \xrightarrow{a} s'\} = \emptyset$, and player $\exists$ wins iff $C_n = s \vdash [a]\,\psi$ and $\{s' : s \xrightarrow{a} s'\} = \emptyset$. Since winning conditions for infinite plays do not depend on modalities, they remain the same. Furthermore, the game rules for modal operators reduce to:

$$(\langle\,\rangle)\ \ \frac{s \vdash \langle a\rangle\phi}{s' \vdash \phi}\ \ [\exists]a:\ \ s \xrightarrow{a} s' \qquad\qquad ([\,])\ \ \frac{s \vdash [a]\,\phi}{s' \vdash \phi}\ \ [\forall]a:\ \ s \xrightarrow{a} s'$$

Clearly, the games just defined are equivalent to the ones presented in [32]. The reason for this coincidence is that when a modality $\langle a\rangle\phi$ (resp. $[a]\,\phi$) is encountered, only player $\exists$ (resp. player $\forall$) gets to choose both the next subformula and the transition used to verify (resp. falsify) the truth value of ϕ.

Now, let us look at the case when a model with an empty independence relation is considered. In such a case the rules $([\,]_{nc})$ and $(\bowtie)$ become trivially true and $(\langle\,\rangle_{nc})$ and $(*)$ trivially false since in an interleaving model all pairs of transitions are in $\leq$. For these reasons the elements that belong to the sets $\mathfrak{P}$ and $\mathfrak{A}$ need no longer be considered and the rules $([\,]_c)$ and $(\langle\,\rangle_c)$ become $([\,])$ and $(\langle\,\rangle)$, respectively. The other rules remain the same. □

7. Model-Checking Partial Order Models of Concurrency

In this section we use trace LMSO model-checking games to push forward the decidability border of the model-checking problem of a particular class of partial order models, namely, of a class of event structures [26, 34]. More precisely, we improve previous results [21, 27] in terms of temporal expressive power.

7.1. SFL on Trace Event Structures

As we have shown in the previous sections, trace LMSO model-checking games can be played in either finite or infinite state systems (with finite branching). However, decidability for the games was proved only for finite systems. Therefore, if the system at hand has recursive behaviour and, moreover, is represented by an event structure, then the TSI representation of it may be infinite, and decidability is not guaranteed.

We now analyse the decidability of trace LMSO model-checking games for a special class of infinite, but regular, event structures called *regular trace event structures*. This class of systems was introduced in [34] by Thiagarajan in order to give a canonical representation to the set of Mazurkiewicz traces modelling the behaviour of a finite concurrent system. The model-checking problem for this class of models has been studied elsewhere [21, 27], and shown to be rather difficult. In the reminder of this section we show that model-checking SFL properties of this kind of systems is also decidable.

As shown in Section 2, an event structure $\mathfrak{E} = (E, \preccurlyeq, \sharp, \eta, \Sigma)$ determines a TSI model $\mathfrak{T} = (S, s_0, T, I, \Sigma)$ by means of an inclusion functor from the category $\mathcal{ES}$ of event structures to the category $\mathcal{TSI}$ of TSI. The mapping we presented in Section 2 was given in a set-theoretic way since such a presentation is more convenient for us. A categorical one can be found in [18]. Let $\lambda : \mathcal{ES} \to \mathcal{TSI}$ be such a construction.

Definition 7.1. (Regular trace event structures) A *regular trace event structure* is an event structure $\mathfrak{E} = (E, \preccurlyeq, \sharp, \eta, \Sigma)$ as defined before, where for all configurations C of $\mathfrak{E}$, and for all events $e \in C$, the set of future non-isomorphic configurations rooted at e defines an equivalence relation of finite index.

Let *Conf* be the set of *configurations* of $\mathfrak{E}$. Notice that the restriction to image-finite models implies that the partial order $\preccurlyeq$ of $\mathfrak{E}$ is of *finite branching*, and hence for all $C \in Conf$, the set of immediately next configurations is bounded. Also notice that the set of states S of the TSI representation of an event structure $\mathfrak{E}$ is isomorphic to the set *Conf* of configurations of $\mathfrak{E}$.

7.2. A Computable Folding Functor from Event Structures to TSI

In order to overcome the problem of dealing with infinite event structures, such as the regular trace event structures just defined, we present a new morphism (a functor) that folds a possibly infinite event structures into a TSI. This way, a finite process space can be constructed so as to give the semantics of SFL formulae, and hence, play a trace LMSO model-checking game in a finite board. Such a morphism and the procedure to effectively compute it is described below.

The Quotient Set Method. Let $Q = (Conf/\sim)$ be the *quotient set representation* of *Conf* by $\sim$ in a finite or infinite event structure $\mathfrak{E}$, where *Conf* is the set of configurations in $\mathfrak{E}$ and $\sim$ is an equivalence relation on such configurations. The equivalence class $[X]_\sim$ of a configuration $X \in Conf$ is the set $\{C \in Conf \mid C \sim X\}$. A quotient set Q where $\sim$ is decidable is said to have a decidable characteristic function, and will be called a *computable quotient set*.

Definition 7.2. (Regular quotient sets) A *regular quotient set* $(Conf/\sim)$ of an event structure $\mathfrak{E}$ is a computable quotient set representation of $\mathfrak{E}$ with a finite number of equivalence classes.

Having defined a regular quotient set representation of $\mathfrak{E}$, the morphism $\lambda : \mathcal{ES} \to \mathcal{TSI}$ above can be modified to defined a new map $\lambda_f : \mathcal{ES} \to \mathcal{TSI}$ which folds a (possibly infinite) event structure into a TSI:

$$\begin{aligned} S &= \{[C]_\sim \subseteq Conf \mid \exists [X]_\sim \in Q = (Conf/\sim).\ C \sim X\} \\ T &= \{([C]_\sim, a, [C']_\sim) \in S \times \Sigma \times S \mid \exists e \in E.\ \eta(e) = a, e \notin C, C' = C \cup \{e\}\} \\ I &= \{(([C_1]_\sim, a, [C'_1]_\sim), ([C_2]_\sim, b, [C'_2]_\sim)) \in T \times T \mid \exists (e_1, e_2) \in \mathsf{co}. \\ &\quad \eta(e_1) = a, \eta(e_2) = b, C'_1 = C_1 \cup \{e_1\}, C'_2 = C_2 \cup \{e_2\}\} \end{aligned}$$

where the initial state s_0 is the equivalence class $[C]_\sim$ such that $C \sim \{\}$.

Lemma 7.3. *Let $\mathfrak{T}$ be a TSI and $\mathfrak{E}$ an event structure. If $\mathfrak{T} = \lambda_f(\mathfrak{E})$, then the models $(\mathfrak{T}, \mathcal{V})$ and $(\mathfrak{E}, \mathcal{V})$ satisfy the same set of SFL formulae.*

Proof. The morphism $\lambda_f : \mathcal{ES} \to \mathcal{TSI}$ from the category of event structures to the category of TSI has a unique right adjoint $\varepsilon : \mathcal{TSI} \to \mathcal{ES}$, the unfolding functor that preserves labelling and the independence relation between events, such that for any $\mathfrak{E}$ we have that $\mathfrak{E}' = (\varepsilon \circ \lambda_f)\ (\mathfrak{E})$, where $\mathfrak{E}'$ is isomorphic to $\mathfrak{E}$. But SFL formulae do not distinguish between models and their unfoldings, and hence cannot distinguish between $(\mathfrak{T}, \mathcal{V})$ and $(\mathfrak{E}', \mathcal{V})$. Moreover, SFL formulae do not distinguish between isomorphic models equally labelled, and therefore cannot distinguish between $(\mathfrak{E}', \mathcal{V})$ and $(\mathfrak{E}, \mathcal{V})$ either. □

Having defined a morphism λ_f that preserves SFL properties, one can now define a procedure that constructs a TSI model from a given event structure.

Definition 7.4. (Representative sets) Let $\mathfrak{E} = (E, \preccurlyeq, \sharp, \eta, \Sigma)$ be an event structure and $(Conf/\sim)$ a regular quotient set representation of $\mathfrak{E}$. A *representative set* E_r of $\mathfrak{E}$ is a subset of E such that $\forall C \in Conf.\ \exists X \subseteq E_r.\ C \sim X$.

Lemma 7.5. *Let $\mathfrak{E}$ be an event structure. If $\mathfrak{E}$ is represented as a regular quotient set $(Conf/\sim)$, then a finite representative set E_r of $\mathfrak{E}$ is effectively computable.*

Proof. Construct a finite representative set E_r as follows. Start with $E_r = \emptyset$ and $C_j = C_0 = \emptyset$, the initial configuration or root of the event structure. Check $C_j \sim X_i$ for every equivalence class $[X_i]_\sim$ in $Q = (Conf/\sim)$ and whenever $C_j \sim X_i$ holds define both a new quotient set $Q' = Q \setminus [X_i]_\sim$ and a new $E_r = E_r \cup C_j$. This subprocedure terminates because there are only finitely many equivalence classes to check and the characteristic function of the quotient set is decidable. Now, do this recursively in a breadth-first search fashion in the partial order defined on E by $\preccurlyeq$, and stop when the quotient set is empty. Since $\preccurlyeq$ is of finite branching and all equivalence classes must have finite configurations, the procedure is bounded both in depth and breath and the quotient set will always eventually get smaller. Hence, such a procedure always terminates. It is easy to see that this procedure only terminates when E_r is a representative set of $\mathfrak{E}$. □

A finite representative set E_r is big enough to define all states in the TSI representation of $\mathfrak{E}$ when using λ_f. However, such a set may not be enough to recognize all transitions in the TSI. In particular, cycles cannot be recognized using E_r. Therefore, it is necessary to compute a set E_f where cycles in the TSI can be recognized. We call E_f a *complete representative set* of $\mathfrak{E}$. The procedure to construct E_f is similar to the previous one.

Lemma 7.6. *Let $\mathfrak{E} = (E, \preccurlyeq, \sharp, \eta, \Sigma)$ be an event structure and E_r a finite representative set of $\mathfrak{E}$. If $\mathfrak{E}$ is represented as a regular quotient set $(Conf/\sim)$, then a finite complete representative set E_f of $\mathfrak{E}$ is effectively computable.*

Proof. Start with $E_f = E_r$, and set $\mathfrak{C} = Conf(E_r)$, the set of configurations generated by E_r. For each C_j in E_r check in $\preccurlyeq$ the set $Next(C_j)$ of next configurations to C_j, i.e., those configurations C'_j such that $C'_j = C_j \cup \{e\}$ for some event e in $E \setminus C_j$. Having computed $Next(C_j)$, set $E_f = E_f \cup (\bigcup Next(C_j))$ and $\mathfrak{C} = \mathfrak{C} \setminus \{C_j\}$, and stop when $\mathfrak{C}$ is empty. This procedure behaves as the one described previously. Notice that at the end of this procedure E_f is complete since it contains the next configurations of all elements in E_r. □

Proposition 7.7. *The TSI $\mathfrak{T}$ generated from an event structure $\mathfrak{E}$ using λ_f and a finite complete representative E_f of $\mathfrak{E}$ is the smallest TSI that represents $\mathfrak{E}$.*

Proof. From Lemmas 7.5 and 7.6. There is only one state in $\mathfrak{T}$ for each equivalence class in the quotient set representation of $\mathfrak{E}$. Similarly there can be only one transition in $\mathfrak{T}$ for each relation on the equivalence classes of configurations in $\mathfrak{E}$ since, due to **A1** of TSI (determinacy), λ_f forgets repeated transitions in T. □

7.3. Temporal Verification of Regular Infinite Event Structures

Based on Lemmas 7.3 and 7.6 and on Theorem 6.3, we can give a decidability result for the class of event structures studied in [21, 34] against SFL specifications. Such a result, which is obtained by representing a regular event structure as a regular quotient set, is a corollary of the following theorem:

Theorem 7.8. *The model-checking problem for an event structure $\mathfrak{E}$ represented as a regular quotient set $(Conf/\sim)$ against SFL specifications is decidable.*

Proof. Due to Lemma 7.6 one can construct a finite complete representative set E_f of E. Then a finite TSI $\mathfrak{T}$ that satisfies the same set of SFL formulae as $\mathfrak{E}$ can be defined by using the folding map λ_f from event structures to TSI, and using E_f instead of E as the new set of events. Since such a morphism preserves all SFL properties (Lemma 7.3), the model-checking problem for this kind of event structures can be reduced to solving the model-checking game for finite TSI, and hence for finite systems in general, which due to Theorem 6.3 is decidable. □

7.3.1. Regular Event Structures as Finite CCS Processes.

A regular event structure can be generated by a finite concurrent system represented by a finite number of (possibly recursive) CCS processes [23, 36]. Syntactic restrictions on CCS that generate only finite systems have been studied. Notice that the combination of the *syntactic* restriction to finite CCS processes and the *semantic* restriction to image-finite models give the requirements for regularity on the event structures that are generated, in particular, of the regular trace event structures defined before.

Now, w.l.o.g., consider only deterministic CCS processes without auto-concurrency. A CCS process is deterministic if whenever $a.M + b.N$, then $a \neq b$, and similarly has no auto-concurrency if whenever $a.M \parallel b.N$, then $a \neq b$. Notice that any CCS process P that either is nondeterministic or has auto-concurrency can be converted into an equivalent process Q which generates an event structure that is isomorphic, up to relabelling of events, to the one generated by P.

Eliminating nondeterminism and auto-concurrency can be done by relabelling events in $\mathcal{P}(P)$, the powerset of CCS processes of P, with an injective map $\theta : \Sigma \to \Sigma^*$ (where Σ^* is a set of labels

and $\Sigma \subseteq \Sigma^*$), and by extending the Synchronization Algebra [36] according to the new labelling of events so as to preserve pairs of (labels of) events that can synchronize. Also notice that the original labelling can always be recovered from the new one, i.e., the one associated with the event structure generated by Q, since θ is injective and hence has inverse $\theta^{-1} : \Sigma^* \to \Sigma$.

7.3.2. Finite CCS Processes as Regular Quotient Sets.

Call $ES\,Proc(P)$ the set of configurations of the event structure generated by a CCS process P of the kind described above. The set $ES\,Proc(P)$ together with an equivalence relation between CCS processes $\equiv_{CCS}$ given simply by syntactic equality between them is a regular quotient set representation ($ES\,Proc(P)$ / $\equiv_{CCS}$) of the event structure generated by P.

Notice that since there are finitely many different CCS expressions, i.e., $\mathcal{P}(P)$ is finite, then the event structure generated by P is of finite-branching and the number of equivalence classes is also bounded. Finally, $\equiv_{CCS}$ is clearly decidable because the process P is always associated with the $\emptyset$ configuration and any other configuration in $ES\,Proc(P)$ can be associated with only one CCS expression in $\mathcal{P}(P)$ as they are deterministic and have no auto-concurrency after relabelling.

The previous simple observations lead to the following result:

Corollary 7.9. *Model-checking regular trace event structures against Separation Fixpoint Logic specifications is decidable.*

8. Discussion and Related Work

Model-checking games have been an active area of research in the last decades (cf. [12, 35]). They have been studied from both theoretical and practical perspectives. For instance, for the proper definition of their mathematical properties [13, 20], or for the construction of tools for property verification [31]. Most approaches based on games have considered either only interleaving systems or the one-step interleaving semantics of partial order models. Our work differs from these approaches in that we deal with games played on partial order models without considering interleaving simplifications. Although verification procedures in finite partial order models can be undecidable, the game presented here is *decidable* in the finite case.

Regarding model-checking in a broader sense, many procedures, not only game-theoretic, have been studied elsewhere for concurrent systems both with interleaving and with partial order semantics. See, e.g., [4, 28] and the references therein for several examples of various techniques and approaches to model-checking concurrent systems. However, since our main motivation was to develop a decision procedure to verify concurrent systems with partial order models, only the techniques considering these kinds of systems relate to our work, though, as said before, such procedures are not game-theoretic.

Regarding the temporal verification of event structures, previous studies have been done on restricted classes. Closer to our work is [21, 27]. Indeed, model-checking regular trace event structures has turned out to be rather difficult and previous work has shown that verifying MSO properties on these structures is already *undecidable*. For this reason weaker logics have been studied. Unfortunately, although very interesting results have been achieved, especially in [21] where CTL* properties can be verified, previous approaches have not managed to define decidable theories for a logic with enough power to express all usual temporal properties as can be done with L_μ in the interleaving case, and hence with SFL in a partial order setting.

Recall that one of the reasons why L_μ is more expressive than CTL* is that L_μ can express properties about "moments" in computation paths, whereas CTL* in general cannot do so. Similarly, one can think of simple properties that talk about moments in traces of partial order models. Those kinds of properties are not expressible in logics whose temporal expressive power equals that of CTL* on interleaving models, e.g., [24]. For instance, the following temporal property would not be expressible: "along any *trace*, at all even moments ϕ holds, and at all odd moments ϕ may hold or not", which is the partial order version of the same property for paths, or in general the computations of an interleaving system (cf. [5]). This means that there are temporal properties of partial order models expressible with SFL formulae which are not definable with other logics (over partial order models) whose temporal expressive power on traces equals that of CTL* on trees, labelled transition graphs, or Kripke structures.

Finally, the difference between [21] and the approach we presented here is that in [21] a *global* second-order quantification on conflict-free sets in the partial order is permitted, whereas only a *local* second-order quantification in the same kind of sets is defined here, but such a second-order power can be embedded into fixpoint specifications, which in turn allows one to express more temporal properties. Therefore, we have improved in terms of temporal expressive power previous results on model-checking regular trace event structures against a branching-time logic. Our work is the first (local) game approach in doing so.

9. Conclusion

In this paper we introduced a new kind of model-checking games where both players are allowed to choose *sets* of independent elements in the underlying model. These games, which we call trace LMSO model-checking games, are proved to be *sound* and *complete*, and therefore determined. They can be played on partial order models of concurrency since the one-step interleaving semantics of such models need not be considered.

However, the results of this work suggest that there may be a general approach to verification, since we have actually defined a *uniform* framework for model-checking several different kinds of concurrent systems, not only those with partial order semantics, since interleaving systems appear as a special case of our framework. This is clearly reflected by the fact that we got *for free* the local model-checking procedure for interleaving systems defined by Stirling for L_μ.

We also showed that by defining infinite games where both players have a *local* second-order power on *conflict-free* sets of transitions, i.e., those in the same *trace*, one can obtain new positive decidability results on the study of partial order models of concurrency. Indeed, we have pushed forward the borderline of the decidability of model-checking event structures. To the best of our knowledge the technique we presented here is the only game-based procedure defined so far that can be used to verify all usual temporal properties of the kind of event structures we studied. We wonder how much further one can go in terms of temporal expressive power before reaching the MSO undecidability barrier when model-checking event structures.

Acknowledgements.

We thank the reviewers of the Information and Computation journal and of CONCUR 2009 for their comments, respectively, on this article and on a preliminary conference version [15]. The first author was financially supported by an Overseas Research Studentship (ORS) award and a School of Informatics PhD studentship at The University of Edinburgh.

References

[1] S. Abramsky, P.A. Melliès, Concurrent games and full completeness, in: LICS, IEEE Computer Society, 1999, pp. 431–442.

[2] J.v. Benthem, Logic games, from tools to models of interaction, in: R.P. A. Gupta, J.v. Benthem (Eds.), Logic at the Crossroads, Allied Publishers, 2007, pp. 283–317.

[3] J.C. Bradfield, C. Stirling, Modal mu-calculi, in: P. Blackburn, J. van Benthem, F. Wolter (Eds.), Handbook of Modal Logic, Elsevier, 2007, pp. 721–756.

[4] E. Clarke, O. Grumberg, D. Peled, Model checking, The MIT Press, 1999.

[5] M. Dam, CTL* and ECTL* as fragments of the modal mu-calculus, Theor. Comput. Sci. 126 (1994) 77–96.

[6] P. Degano, R.D. Nicola, U. Montanari, A distributed operational semantics for CCS based on condition/event systems, Acta Inf. 26 (1988) 59–91.

[7] M. Droste, Concurrent automata and domains, Int. J. Found. Comput. Sci. 3 (1992) 389–418.

[8] J. Esparza, K. Heljanko, Unfoldings - A partial-order approach to model checking, EATCS Monographs in Theoretical Computer Science, Springer, 2008.

[9] R. Gerth, R. Kuiper, D. Peled, W. Penczek, A partial order approach to branching time logic model checking, Inf. Comput. 150 (1999) 132–152.

[10] R.J.v. Glabbeek, U. Goltz, Refinement of actions and equivalence notions for concurrent systems, Acta Inf. 37 (2001) 229–327.

[11] P. Godefroid, P. Wolper, A partial approach to model checking, Inf. Comput. 110 (1994) 305–326.

[12] E. Grädel, Model checking games, Electr. Notes Theor. Comput. Sci. 67 (2002).

[13] E. Grädel, W. Thomas, T. Wilke (Eds.), Automata, logics, and infinite games, LNCS **2500**, Springer, 2002.

[14] J. Gutierrez, Logics and bisimulation games for concurrency, causality and conflict, in: L. de Alfaro (Ed.), FOSSACS, LNCS **5504**, Springer, 2009, pp. 48–62.

[15] J. Gutierrez, J.C. Bradfield, Model-checking games for fixpoint logics with partial order models, in: M. Bravetti, G. Zavattaro (Eds.), CONCUR, LNCS **5710**, Springer, 2009, pp. 354–368.

[16] M. Hennessy, R. Milner, Algebraic laws for nondeterminism and concurrency, J. ACM 32 (1985) 137–161.

[17] T.T. Hildebrandt, V. Sassone, Comparing transition systems with independence and asynchronous transition systems, in: U. Montanari, V. Sassone (Eds.), CONCUR, LNCS **1119**, Springer, 1996, pp. 84–97.

[18] A. Joyal, M. Nielsen, G. Winskel, Bisimulation from open maps, Inf. Comput. 127 (1996) 164–185.

[19] D. Kozen, Results on the propositional mu-calculus, Theor. Comput. Sci. 27 (1983) 333–354.

[20] M. Lange, Games for modal and temporal logics, Ph.D. thesis, The University of Edinburgh, 2002.

[21] P. Madhusudan, Model-checking trace event structures, in: LICS, IEEE Computer Society, 2003, pp. 371–380.

[22] A.W. Mazurkiewicz, Introduction to trace theory, in: V. Diekert, G. Rozenberg (Eds.), The Book of Traces, World Scientific, 1995, pp. 3–42.

[23] R. Milner, Communication and concurrency, Prentice-Hall, 1989.

[24] F. Moller, A.M. Rabinovich, On the expressive power of CTL*, in: LICS, IEEE Computer Society, 1999, pp. 360–369.

[25] M. Nielsen, G.D. Plotkin, G. Winskel, Petri nets, event structures and domains, Part I, Theor. Comput. Sci. 13 (1981) 85–108.

[26] M. Nielsen, G. Winskel, Models for concurrency, in: Handbook of Logic in Computer Science, Oxford University Press, 1995, pp. 1–148.

[27] W. Penczek, Model-checking for a subclass of event structures, in: E. Brinksma (Ed.), TACAS, LNCS **1217**, Springer, 1997, pp. 145–164.

[28] W. Penczek, R. Kuiper, Traces and logic, in: V. Diekert, G. Rozenberg (Eds.), The Book of Traces, World Scientific, 1995, pp. 307–390.

[29] J. Reynolds, Separation logic: A logic for shared mutable data structures, in: LICS, IEEE Computer Society, 2002, pp. 55–74.

[30] V. Sassone, M. Nielsen, G. Winskel, Models for concurrency: Towards a classification, Theor. Comput. Sci. 170 (1996) 297–348.

[31] P. Stevens, C. Stirling, Practical model-checking using games, in: B. Steffen (Ed.), TACAS, LNCS **1384**, Springer, 1998, pp. 85–101.

[32] C. Stirling, Local model checking games, in: I. Lee, S.A. Smolka (Eds.), CONCUR, LNCS **962**, Springer, 1995, pp. 1–11.

[33] C. Stirling, Modal and temporal properties of processes, Texts in Computer Science, Springer, 2001.

[34] P.S. Thiagarajan, Regular trace event structures, Technical Report, BRICS, 1996.

[35] I. Walukiewicz, A landscape with games in the background, in: LICS, IEEE Computer Society, 2004, pp. 356–366.

[36] G. Winskel, Event structure semantics for CCS and related languages, in: M. Nielsen, E. Schmidt (Eds.), ICALP, LNCS **140**, Springer, 1982, pp. 561–576.