Theoroidal Maps as Algebraic Simulations[⋆]

Narciso Martí-Oliet[1], José Meseguer[2], and Miguel Palomino[1]

[1] Departamento de Sistemas Informáticos, Universidad Complutense de Madrid
[2] Computer Science Department, University of Illinois at Urbana-Champaign
{narciso, miguelpt}@sip.ucm.es
meseguer@cs.uiuc.edu

Abstract. Computational systems are often represented by means of Kripke structures, and related using simulations. We propose rewriting logic as a flexible and executable framework in which to formally specify these mathematical models, and introduce a particular and elegant way of representing simulations in it: theoroidal maps. A categorical viewpoint is very natural in the study of these structures and we show how to organize Kripke structures in categories that afterwards are lifted to the rewriting logic's level. We illustrate the use of theoroidal maps with two applications: predicate abstraction and the study of fairness constraints.

1 Introduction

Formal reasoning about concurrent systems typically involves two levels of specification: (1) a *system specification* level, in which an explicit computational description of a concurrent system is given; and (2) a *property specification* level, in which different safety and liveness properties satisfied by the system are specified. A system specification typically determines a *mathematical model* (or set of models) about which we want to verify that some properties are satisfied. Frequently used mathematical models include transition systems, and Kripke structures—i.e., transition systems decorated with information about satisfaction of atomic predicates. For properties, different temporal and modal logics can be used; CTL* [5] is a common choice, because it contains the widely used LTL and CTL logics as special cases.

But how can such mathematical models be formally specified? There are many possibilities. In this paper we specify them by means of *rewrite theories*. This is a natural choice, because rewriting logic provides a flexible framework for specifying a wide range of concurrent systems at a high level [16, 14], yet in an executable way supported by languages such as Maude in which we can simulate and model check such systems [7, 8]. Essentially, system states are specified as elements of an initial algebra, and (parameterized) transitions as rewrite rules. Furthermore, it is then very easy to

[⋆] Research supported by ONR Grant N00014-02-1-0715, NSF Grant CCR-0234524, and by DARPA through Air Force Research Laboratory Contract F30602-02-C-0130; and by the Spanish projects MELODIAS TIC 2002-01167 and MIDAS TIC 2003–0100.

equationally specify *atomic predicates* holding on the states in a theory extension. In this way, we can associate a Kripke structure $\mathscr{K}(\mathscr{R},k)_{\Pi}$ to a rewrite theory $\mathscr{R}$, a kind of states k, and atomic propositions Π. Given a CTL* formula φ, then the issue of whether the system specification satisfies the property φ becomes the question of verifying whether $\mathscr{K}(\mathscr{R},k)_{\Pi} \models \varphi$ holds.

However, it may be considerably easier to verify such a satisfaction relation using a *different* system specification $\mathscr{R}'$. For example, $\mathscr{K}(\mathscr{R},k)_{\Pi}$ may have infinitely many states, whereas $\mathscr{K}(\mathscr{R}',k)_{\Pi}$ may be a finite-state abstraction of $\mathscr{R}$ [19], so that we can use a model checker to verify $\mathscr{K}(\mathscr{R}',k)_{\Pi} \models \varphi$. From this we can infer that $\mathscr{K}(\mathscr{R},k)_{\Pi} \models \varphi$ holds, provided that $\mathscr{R}$ and $\mathscr{R}'$ can be related by an adequate *simulation map* $H : \mathscr{R} \longrightarrow \mathscr{R}'$. This of course suggests a categorical approach, and also exploring an adequate notion of theory morphism to define such simulations at a logical level. This is the goal of this paper. Specifically we:

- Define a category with objects Kripke structures and morphisms quite general "stuttering simulations," and show that properties specified by a natural subclass of CTL* formulas are reflected by such simulations.
- Show that those CTL* formulas, with Kripke structures as models and simulations as morphisms, form an *institution* [12].
- Explain the $\mathscr{K}(\mathscr{R},k)_{\Pi}$ construction in detail allowing us to specify Kripke structures by means of rewrite theories.
- Present a new notion of *partial theory morphism* which allows a more general and expressive way of relating theories than with ordinary theory morphisms.
- Define a category with rewrite theories (plus the specification of the kind of states and the state predicates) as objects, and suitable partial theory morphisms as morphisms, and show that they define a useful class of simulations between the underlying Kripke structures, which we call *theoroidal simulations*.
- Illustrate the usefulness of this notion in several areas, including predicate abstraction, and reasoning about temporal logic properties under fairness assumptions. Furthermore, theoroidal simulations greatly generalize *equational abstractions*, which were already shown to be very useful in [19].

An extended version of this paper with the missing proofs can be found in [15].

2 Prerequisites

2.1 Computational Systems

When reasoning about computational systems, it is usually convenient to abstract from as many details as possible by means of simple mathematical models that can be used to reason about them. For a state-based system we can represent its behavior by means of a *transition system*, which is a pair $\mathscr{A} = (A, \rightarrow_{\mathscr{A}})$ with A a set of states and $\rightarrow_{\mathscr{A}} \subseteq A \times A$ a binary relation called the transition relation.

A transition system, however, does not include any information about the relevant properties of the system. In order to reason about such properties it is necessary to add information about the atomic properties that hold in each state. In what follows, we assume a fixed set AP of atomic propositions and define a *Kripke structure* as a triple $\mathscr{A} = (A, \rightarrow_{\mathscr{A}}, L_{\mathscr{A}})$, where $(A, \rightarrow_{\mathscr{A}})$ is a transition system with $\rightarrow_{\mathscr{A}}$ a *total* relation, and $L_{\mathscr{A}} : A \rightarrow \mathscr{P}(AP)$ is a labeling function associating to each state the set of atomic propositions that hold in it. Note that the transition relation must be total [5]; given an arbitrary relation $\rightarrow$, we write $\rightarrow^{\bullet}$ for the total relation that extends $\rightarrow$ by adding a pair $a \rightarrow^{\bullet} a$ for each a such that there is no b with $a \rightarrow b$. A path in $\mathscr{A}$ is a function $\pi : \mathbb{N} \longrightarrow A$ such that, for each $i \in \mathbb{N}$, $\pi(i) \rightarrow_{\mathscr{A}} \pi(i+1)$.

To specify system properties we will use the logic ACTL$^*(AP)$, which is a sublogic of the branching-time temporal logic CTL$^*(AP)$ (see for example [5–Sect. 3.1]). There are two types of formulas in CTL$^*(AP)$: state formulas, denoted by State(AP), and path formulas, denoted by Path(AP). The semantics of the logic, specifying the satisfaction relations $\mathscr{A}, a \models \varphi$ and $\mathscr{A}, \pi \models \psi$ for a Kripke structure $\mathscr{A}$, an initial state $a \in A$, a state formula φ, a path π, and a path formula ψ, is defined as usual [5]. ACTL$^*(AP)$ is the restriction of CTL$^*(AP)$ to those formulas such that their negation-normal forms (with negations pushed to atoms) do not contain any existential path quantifiers. Sometimes, to avoid introducing existential quantifiers implicitly, it is more convenient to restrict ourselves to the negation-free fragment ACTL$^*\backslash\neg(AP)$ of ACTL$^*(AP)$, defined as follows:[1]

$$\begin{array}{ll} \text{state formulas:} & \varphi = p \in AP \mid \top \mid \bot \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{A}\psi \\ \text{path formulas:} & \psi = \varphi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \psi\mathbf{U}\psi \mid \psi\mathbf{R}\psi \mid \mathbf{G}\psi \mid \mathbf{F}\psi. \end{array}$$

We write State$\backslash\neg(AP)$ and Path$\backslash\neg(AP)$ for the sets of state and path formulas in ACTL$^*\backslash\neg(AP)$, respectively.

2.2 Rewriting Logic

Rewriting logic [16] provides a very flexible framework for the system-level specification of concurrent systems. It is parameterized by an underlying equational logic, which we will also use to specify the system's properties; in this paper we use membership equational logic [17], whose main features we now review.

A *signature* in membership equational logic is a triple (K, Σ, S) (just Σ in the following), with K a set of *kinds*, $\Sigma = \{\Sigma_{k_1 \ldots k_n, k}\}_{(k_1 \ldots k_n, k) \in K^* \times K}$ a many-kinded signature, and $S = \{S_k\}_{k \in K}$ a pairwise disjoint K-kinded family of sets of *sorts*. The kind of a sort s is denoted by $[s]$. We write $T_{\Sigma,k}$ and $T_{\Sigma,k}(X)$ to denote respectively the set of ground Σ-terms with kind k and of Σ-terms with kind k over variables in X, where $X = \{x_1 : k_1, \ldots, x_n : k_n\}$ is a set of K-kinded variables. Intuitively, terms with a kind but without a sort represent undefined or error elements. An atomic formula is either an *equation* $t = t'$, where t and t' are Σ-terms of the same kind, or a *membership assertion* of the form $t : s$, where the term t has kind k and $s \in S_k$. Sentences are conditional formulas of the form $(\forall X) A_0$ **if** $A_1 \wedge \ldots \wedge A_n$, where each A_i is either an equation or a membership assertion, and X is a set of K-kinded variables containing all the variables in the A_i. A theory is a pair (Σ, E), where E is a set of sentences in membership equational logic over the signature Σ. We write $(\Sigma, E) \vdash \phi$, or just $E \vdash \phi$ if Σ is clear from the context, to

[1] **X**, **G**, and **F** stand for the classic *next* ($\bigcirc$), *henceforth* ($\Box$), and *eventually* ($\Diamond$) LTL operators.

denote that (Σ, E) entails the sentence ϕ in the proof system of membership equational logic [17]. A theory (Σ, E) has an initial model $T_{\Sigma/E}$ whose elements are E-equivalence classes of terms $[t]$. Algebras over a signature are defined in a standard manner; we denote by A_f the interpretation of an operator f in the algebra A and by A_t that of a term t, and refer to [17] for a detailed presentation of the model theory.

Concurrent systems are axiomatized in rewriting logic by means of *rewrite theories* [16] of the form $\mathscr{R} = (\Sigma, E, R)$. The set of states is described by a membership equational theory (Σ, E) as the algebraic data type $T_{\Sigma/E,k}$ associated to the initial algebra $T_{\Sigma/E}$ of (Σ, E) by the choice of a kind k of states in Σ. The system's *transitions* are axiomatized by the *conditional rewrite rules R* which are of the form

$$\lambda : (\forall X)\, t \longrightarrow t' \textbf{ if } \bigwedge_{i \in I} p_i = q_i \wedge \bigwedge_{j \in J} w_j : s_j \wedge \bigwedge_{l \in L} t_l \longrightarrow t'_l\,,$$

with λ a label, $p_i = q_i$ and $w_j : s_j$ atomic formulas in membership equational logic for $i \in I$ and $j \in J$, and for appropriate kinds k and k_l, $t, t' \in T_{\Sigma,k}(X)$, and $t_l, t'_l \in T_{\Sigma,k_l}(X)$ for $l \in L$. Under reasonable assumptions about E and R, rewrite theories are *executable*. Indeed, there are several rewriting logic language implementations, including CafeOBJ [11], ELAN [3], and Maude [7, 8]. Rewriting logic then has inference rules to infer all the possible concurrent computations in a system [16, 4], in the sense that, given two states $[u], [v] \in T_{\Sigma/E,k}$, we can reach $[v]$ from $[u]$ by some possibly complex concurrent computation iff we can prove $u \longrightarrow v$ in the logic; we denote this provability by $\mathscr{R} \vdash u \longrightarrow v$. In particular we can easily define the *one-step $\mathscr{R}$-rewriting relation*, which is a binary relation $\rightarrow^1_{\mathscr{R},k}$ on $T_{\Sigma,k}$ that holds between terms $u, v \in T_{\Sigma,k}$ iff there is a proof of $u \longrightarrow v$ in which only one rewrite rule in R is applied to a single subterm.

2.3 Computational Systems in Rewriting Logic

To associate a transition system to a rewrite theory we transfer the one-step rewriting relation $\rightarrow^1_{\mathscr{R},k}$ from terms in $T_{\Sigma,k}$ to states in $T_{\Sigma/E,k}$, by defining $[u] \rightarrow^1_{\mathscr{R},k} [v]$ iff $u' \rightarrow^1_{\mathscr{R},k} v'$ for some $u' \in [u]$, $v' \in [v]$. This definition determines a transition system $\mathscr{T}(\mathscr{R})_k = (T_{\Sigma/E,k}, (\rightarrow^1_{\mathscr{R},k})^\bullet)$ for each $k \in K$.

In order to associate temporal properties to a rewrite theory $\mathscr{R} = (\Sigma, E, R)$ we need to make explicit two things: the intended *kind k* of states in the signature Σ, and the relevant *state predicates*. Once the kind k is fixed, the transitions between states are given by $\mathscr{T}(\mathscr{R})_k$. In general, however, the state predicates need not be part of the system specification but only of the property specification. We assume that they have been defined by means of equations D in a *protecting* theory extension $(\Sigma', E \cup D)$ of (Σ, E); that is, the extension is conservative in the sense that the unique Σ-homomorphism $T_{\Sigma/E} \longrightarrow T_{\Sigma'/E \cup D}|_\Sigma$ should be bijective at each sort in Σ. We also assume that $(\Sigma', E \cup D)$ is a protecting theory extension of *BOOL*, the theory of Boolean values. Furthermore, we assume that the syntax defining the state predicates consists of a subsignature $\Pi \subseteq \Sigma'$ of operators, with each $p \in \Pi$ a state predicate symbol that can be *parameterized*, that is, p need not be a constant, but can in general be an operator $p : s_1 \ldots s_n \longrightarrow \mathit{Prop}$, with *Prop* the kind of propositions. If k is the kind of states, the *semantics* of the state predicates Π is defined with the help of an operator $\_ \models \_ : k\ \mathit{Prop} \longrightarrow \mathit{Bool}$ in Σ' and by equations $E \cup D$. By definition, given ground terms $u_1, \ldots, u_n$, we say that the state predicate $p(u_1, \ldots, u_n)$ *holds* in the state $[t]$ iff $E \cup D \vdash t \models p(u_1, \ldots, u_n) = \mathit{true}$.

Then, we associate to a rewrite theory $\mathscr{R} = (\Sigma, E, R)$ (with a selected kind k of states and with state predicates Π) a Kripke structure whose atomic propositions are specified by the set $AP_\Pi = \{\theta(p) \mid p \in \Pi,\ \theta \text{ ground substitution}\}$, where by convention we use the simplified notation $\theta(p)$ to denote the ground term $\theta(p(x_1,\ldots,x_n))$. We define $\mathscr{K}(\mathscr{R},k)_\Pi = (T_{\Sigma/E,k}, (\rightarrow^1_{\mathscr{R},k})^\bullet, L_\Pi)$, where $L_\Pi([t]) = \{\theta(p) \in AP_\Pi \mid \theta(p) \text{ holds in } [t]\}$.

3 Relating Systems

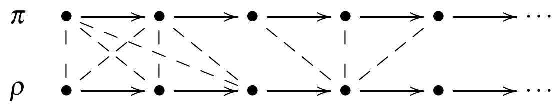
So far we have discussed how to mathematically capture the essential characteristics of computational systems and have proposed rewriting logic as a flexible framework in which to represent them. But we are not interested in computational systems in isolation. We would like to be able to study, for example, if a particular system is an abstraction, or an implementation, of another one. To do that, the concept of *simulation* is introduced.

3.1 Stuttering Simulations

Classically, a simulation $H : \mathscr{A} \longrightarrow \mathscr{B}$ of Kripke structures relates states that satisfy the same atomic propositions in such a way that to every path in $\mathscr{A}$ corresponds a path in $\mathscr{B}$. A key fact is that then every ACTL* formula that holds in $\mathscr{B}$ is also true in $\mathscr{A}$.

Our aim is to generalize the notion of simulation to give it a wider applicability. This generalization should satisfy the same two key properties of basic simulations: (i) be *compositional*, and (ii) *reflect* interesting properties. We achieve this goal by slightly restricting the logic; on the one hand, by forbidding negations (no real expressive power is lost) the condition that related states have to satisfy the same properties can be relaxed, and on the other, by forbidding the *next* operator **X** (see Section 3.2), we can allow paths to be simulated up to *stuttering* (which is what one really cares about most of the time).

Formally, for $\mathscr{A} = (A, \rightarrow_{\mathscr{A}})$ and $\mathscr{B} = (B, \rightarrow_{\mathscr{B}})$ transition systems and $H \subseteq A \times B$ a relation, we say that a path ρ in $\mathscr{B}$ *H-matches* a path π in $\mathscr{A}$ if there are strictly increasing functions $\alpha, \beta : \mathbb{N} \longrightarrow \mathbb{N}$ with $\alpha(0) = \beta(0) = 0$ such that, for all $i, j, k \in \mathbb{N}$, if $\alpha(i) \leq j < \alpha(i+1)$ and $\beta(i) \leq k < \beta(i+1)$, it holds that $\pi(j) H \rho(k)$. For example, the following diagram shows the beginning of two matching paths, where related elements are joined by dashed lines and $\alpha(0) = \beta(0) = 0$, $\alpha(1) = 2$, $\beta(1) = 3$, $\alpha(2) = 5$.



Definition 1. *Given transition systems $\mathscr{A}$ and $\mathscr{B}$, a* stuttering simulation of transition systems *$H : \mathscr{A} \longrightarrow \mathscr{B}$ is a binary relation $H \subseteq A \times B$ such that if aHb, then for each path π in $\mathscr{A}$ starting at a there is a path ρ in $\mathscr{B}$ starting at b that H-matches π. If H is a function we say that H is a* stuttering map of transition systems. *If both H and H^{-1} are stuttering simulations, then we call H a* stuttering bisimulation.

Given Kripke structures $\mathscr{A} = (A, \rightarrow_{\mathscr{A}}, L_{\mathscr{A}})$ and $\mathscr{B} = (B, \rightarrow_{\mathscr{B}}, L_{\mathscr{B}})$ over AP, a stuttering AP-simulation *$H : \mathscr{A} \longrightarrow \mathscr{B}$ is a stuttering simulation of transition systems $H : (A, \rightarrow_{\mathscr{A}}) \longrightarrow (B, \rightarrow_{\mathscr{B}})$ such that if aHb then $L_{\mathscr{B}}(b) \subseteq L_{\mathscr{A}}(a)$. If H is a function we call H a* stuttering AP-map. *We call H a stuttering AP-bisimulation if H and H^{-1} are stuttering AP-simulations. We call H* strict *if aHb implies $L_{\mathscr{B}}(b) = L_{\mathscr{A}}(a)$.*

3.2 The Temporal Logic Institution

Simulations, as defined above, compose, and it is immediate to check that the identity function $1_{\mathscr{A}} : \mathscr{A} \longrightarrow \mathscr{A}$ is a simulation of transition systems and of Kripke structures. Therefore, transition systems together with their simulations define a category **STSys**, and similarly, for each set AP of atomic propositions there is a category $\mathbf{KSSim}_{AP}$ with a subcategory $\mathbf{KSMap}_{AP}$ of stuttering AP-maps. Note that if H is an isomorphism in $\mathbf{KSSim}_{AP}$ then it must be a map and a bisimulation. Note, finally, that the mapping $(A, \rightarrow_{\mathscr{A}}, L_{\mathscr{A}}) \mapsto (A, \rightarrow_{\mathscr{A}})$ extends to a forgetful functor $\mathbf{TS} : \mathbf{KSSim}_{AP} \longrightarrow \mathbf{STSys}$.

Although the main goal of this paper is the study of simulations and their representation in rewriting logic, we believe that a categorical viewpoint is indeed the most natural to understand these generalized simulations and hence consider worthwhile to devote the rest of this section to present some ideas in that context. In what follows we show how these categories can be neatly organized in an institution [12] for the logic ACTL*. Other institutions for temporal logics are discussed in [1], but their notions of signature morphism and of simulation (which roughly corresponds to our notion of bisimulation map) are more limited. As a side effect, we will also construct a Grothendieck category [20] which will allow us to relate Kripke structures over different sets of atomic propositions, further generalizing the notion of simulation.

Let us first define the category of signatures. A simple option would be to choose sets of atomic propositions as objects and functions between them as arrows, but we are aiming for the most general notion that still reflects satisfaction of suitable formulas.[2] For that, let $\mathrm{State}\backslash\{\neg, \mathbf{X}\} : \mathbf{Set} \longrightarrow \mathbf{Set}$ be the functor mapping a set AP to $\mathrm{State}\backslash\{\neg, \mathbf{X}\}(AP)$, the state formulas in $\mathrm{ACTL}^*\backslash\neg(AP)$ that do not contain the *next* operator $\mathbf{X}$, and a function $\alpha : AP \longrightarrow AP'$ to its homomorphic extension

$$\overline{\alpha} : \mathrm{State}\backslash\{\neg, \mathbf{X}\}(AP) \longrightarrow \mathrm{State}\backslash\{\neg, \mathbf{X}\}(AP') .$$

Then, the triple $\langle \mathrm{State}\backslash\{\neg, \mathbf{X}\}, \eta, \mu \rangle$ is a monad [2], where $\eta : Id_{\mathbf{Set}} \Rightarrow \mathrm{State}\backslash\{\neg, \mathbf{X}\}$ and $\mu : \mathrm{State}\backslash\{\neg, \mathbf{X}\} \circ \mathrm{State}\backslash\{\neg, \mathbf{X}\} \Rightarrow \mathrm{State}\backslash\{\neg, \mathbf{X}\}$ are natural transformations such that $\eta_{AP}(p) = p$ and μ "unwraps" a formula into its basic atomic propositions. Our category of signatures will be $\mathbf{Set}_{\mathrm{State}\backslash\{\neg, \mathbf{X}\}}$, the Kleisli category of the monad; its objects are just sets, and the morphisms $AP \longrightarrow AP'$ are functions $\alpha : AP \longrightarrow \mathrm{State}\backslash\{\neg, \mathbf{X}\}(AP')$.

We also need a notion of a *reduct* of a Kripke structure, inspired by that of the reduct of an algebra. Given a function $\alpha : AP \longrightarrow \mathrm{State}(AP')$ and a Kripke structure $\mathscr{A} = (A, \rightarrow_{\mathscr{A}}, L_{\mathscr{A}})$ over AP', we define the *reduct Kripke structure* $\mathscr{A}|_{\alpha} = (A, \rightarrow_{\mathscr{A}}, L_{\mathscr{A}|_{\alpha}})$ over AP, with labeling function $L_{\mathscr{A}|_{\alpha}}(a) = \{p \in AP \mid \mathscr{A}, a \models \alpha(p)\}$. We can now define the desired institution.

Definition 2. *The institution of Kripke structures,* $\mathscr{I}_{\mathbf{K}} = (\mathbf{Sign}_{\mathbf{K}}, sen_{\mathbf{K}}, \mathbf{Mod}_{\mathbf{K}}, \models)$*, is given by:*

[2] The simpler category, however, gives rise to a semiexact institution, which is not true for the one presented in the text; see [15] for more details.

– $\mathbf{Sign_K} = \mathbf{Set}_{\text{State}\backslash\{\neg,\mathbf{X}\}}$.
– $sen_{\mathbf{K}} : \mathbf{Set}_{\text{State}\backslash\{\neg,\mathbf{X}\}} \longrightarrow \mathbf{Set}$ *is the functor mapping a set AP to* $\text{State}\backslash\{\neg,\mathbf{X}\}(AP)$, *and a function* $\alpha : AP \longrightarrow \text{State}\backslash\{\neg,\mathbf{X}\}(AP')$ *to its homomorphic extension* $\overline{\alpha} : \text{State}\backslash\{\neg,\mathbf{X}\}(AP) \longrightarrow \text{State}\backslash\{\neg,\mathbf{X}\}(AP')$.
– $\mathbf{Mod_K} : \mathbf{Set}_{\text{State}\backslash\{\neg,\mathbf{X}\}} \longrightarrow \mathbf{Cat}^{\text{op}}$ *is given by* $\mathbf{Mod_K}(AP) = \mathbf{KSSim}_{AP}$ *and, for* $\alpha : AP \longrightarrow AP'$ *in* $\mathbf{Set}_{\text{State}\backslash\{\neg,\mathbf{X}\}}$, $\mathbf{Mod_K}(\alpha)(\mathscr{A}) = \mathscr{A}|_\alpha$ *and* $\mathbf{Mod_K}(\alpha)(H) = H$.
– *The satisfaction relation is defined as* $\mathscr{A} \models \varphi$ *iff* $\mathscr{A}, a \models \varphi$ *for all* $a \in A$.

Proposition 1. $\mathscr{I}_{\mathbf{K}}$ *is an institution.*

Now, having defined the indexed category $\mathbf{Mod_K}$ allows us to construct the "flattened" category of Kripke structures over arbitrary sets of atomic propositions. Let us denote with **KSSim** the Grothendieck category [20] corresponding to $\mathbf{Mod_K}$; spelling out the definition, this gives rise to our most general notion of simulation. A *stuttering simulation* $(\alpha, H) : (AP, \mathscr{A}) \longrightarrow (AP', \mathscr{B})$ in **KSSim** between a Kripke structure $\mathscr{A}$ over AP and another $\mathscr{B}$ over AP' consists of a function $\alpha : AP \longrightarrow \text{State}\backslash\{\neg,\mathbf{X}\}(AP')$ together with an AP-simulation $H : \mathscr{A} \longrightarrow \mathscr{B}|_\alpha$. We say that (α, H) *reflects a state formula* φ if whenever aHb and $\mathscr{B}, b \models \overline{\alpha}(\varphi)$, then $\mathscr{A}, a \models \varphi$. Then, not only these generalized simulations still compose but they also reflect suitable ACTL* formulas.

Theorem 1. *Stuttering simulations always reflect satisfaction of* ACTL$^*\backslash\{\neg,\mathbf{X}\}$ *formulas. In addition, strict stuttering simulations also reflect satisfaction of* ACTL$^* \setminus \mathbf{X}$ *formulas.*

Note that by using different types of morphisms between Kripke structures and choosing as sentences those temporal formulas reflected by them, we can get different institutions and Grothendieck categories. For example, if we forget about stuttering and only allow simulations that preserve one-step transitions, and define the category of signatures through a functor State : $\mathbf{Set} \longrightarrow \mathbf{Set}$ mapping AP to State(AP), we get the institution of Kripke structures and classic simulations.

4 Theoroidal Maps

We have already noted that, in order to reason about computational systems, these can be abstractly described by means of transition systems and Kripke structures, and that rewriting logic can be used to specify both kinds of structures, as explained in the previous sections. Our goal now is to study how to relate different rewrite theories and how to lift to this specification level all the previous results about simulations of Kripke structures. For this, we consider four increasingly more general ways of defining simulations for rewrite theories specifying a concurrent system:

1. The easiest way of defining a simulation map for a rewrite theory (Σ, E, R) is by means of an *equational abstraction* [19], which consists in simply adding new equations, say E', to get a quotient system specified by $(\Sigma, E \cup E', R)$.
2. The previous method can be generalized by considering, instead of just theory inclusions $(\Sigma, E) \subseteq (\Sigma, E \cup E')$, arbitrary *theory interpretations* $H : (\Sigma, E) \longrightarrow (\Sigma', E')$ allowing arbitrary transformations on the data representation of states.

3. A third alternative consists in defining a simulation *map* between rewrite theories $\mathscr{R}$ and $\mathscr{R}'$ directly at the level of their associated Kripke structures by means of *equationally defined functions*.
4. Finally, the most general case is obtained by defining arbitrary simulations between rewrite theories $\mathscr{R}$ and $\mathscr{R}'$ by means of *rewrite relations*.

For each of the increasingly more general ways above of defining simulations, there are of course associated *correctness conditions* that must be verified. For equational abstractions they are considered in detail in [19]. Here we study the second case, that we call *theoroidal maps*; although not so general as the last two, there are still many interesting examples that can be explained with them, as we illustrate in Section 5. The remaining cases 3–4 will be treated elsewhere.

4.1 Generalized Theory Morphisms

The first thing to do is to make precise the meaning of *theory interpretation*. The idea is to use the standard concepts of signature and theory morphism. However, as we shall see in some of the examples below, the usual definition of signature morphism is sometimes not expressive enough. For this reason we introduce the following generalization of the concept of signature morphism in which a kind or an operator can be *erased*.

Definition 3. *Given two membership equational signatures $\Sigma = (K, \Sigma, S)$ and $\Sigma' = (K', \Sigma', S')$, a* generalized signature morphism *$H : \Sigma \longrightarrow \Sigma'$ is specified by:*

- *partial functions $H : K \longrightarrow K'$ and $H : S \longrightarrow S'$ such that, for all sorts $s \in \Sigma$, if $H(s)$ is defined so is $H([s])$ and $H([s]) = [H(s)]$.*
- *a partial function H assigning, to each $f \in \Sigma_{k_1 \dots k_n, k}$ such that $H(k)$ is defined, a Σ'-term $H(f)$ of kind $H(k)$ such that $vars(H(f)) \subseteq \{x_{i_1} : H(k_{i_1}), \dots, x_{i_m} : H(k_{i_m})\}$, where $k_{i_1}, \dots, k_{i_m}$ is the (possibly empty) subsequence of $k_1, \dots, k_n$ determined by those k_i such that $H(k_i)$ is defined. Otherwise, if $H(k)$ is undefined, so is $H(f)$.*

All standard constructions and results about signature morphisms apply to these generalized ones as well. Given $H : \Sigma \longrightarrow \Sigma'$ and a Σ'-algebra A, its reduct $U_H(A)$ over Σ is defined by:

- For each kind k, $U_H(A)_k = A_{H(k)}$ if $H(k)$ is defined; otherwise $U_H(A)_k = \{*\}$.
- For each sort s, $U_H(A)_s = A_{H(s)}$ if $H(s)$ is defined; otherwise $U_H(A)_s = \{*\}$.
- For each operator $f : k_1 \dots k_n \longrightarrow k$, if $k_{i_1}, \dots, k_{i_m}$ is the subsequence of those kinds in $k_1, \dots, k_n$ for which H is defined,
$$U_H(A)_f(a_1, \dots, a_n) = A_{H(f)}(a_{i_1}, \dots, a_{i_m})\,;$$
otherwise,
$$U_H(A)_f(a_1, \dots, a_n) = *\,.$$

Given generalized signature morphisms $F : \Sigma \longrightarrow \Sigma'$ and $G : \Sigma' \longrightarrow \Sigma''$, their composition $G \circ F$ is defined for a kind k only if both $F(k)$ and $G(F(k))$ are defined, and then it is $(G \circ F)(k) = G(F(k))$; analogously for a sort s and an operator f.

Generalized signature morphisms can also be extended homomorphically to terms, but note that for t of kind k, if $H(k)$ is not defined then $H(t)$ is not defined either. This translation extends to formulas in the expected way, where by convention $H(t = t') = H(t : s) = \top$ if H is not defined for the kind of t (which is the same as that of t' and s). Our desired general notion of "theory interpretation" is then captured by the following:

Definition 4. *Given two membership equational theories* (Σ, E) *and* (Σ', E')*, a* generalized theory morphism *(resp. a* generalized theory morphism with initial semantics*)* $H : (\Sigma, E) \longrightarrow (\Sigma', E')$ *is a generalized signature morphism* $H : \Sigma \longrightarrow \Sigma'$ *such that for each* $\varphi \in E$, $E' \models H(\varphi)$ *(resp.* $T_{\Sigma'/E'} \models H(\varphi)$*).*

Note that, since $T_{\Sigma'/E'} \models E'$, each generalized theory morphism is a fortiori a generalized theory morphism with initial semantics, but not conversely. For example, if (Σ, E) is the theory with one sort, *Nat*, a binary operator $+$, and the equation $(\forall\{x, y : \mathit{Nat}\})\, x + y = y + x$, (Σ', E') is the usual equational definition of addition in Peano arithmetic, and H is the obvious signature inclusion, then we have $T_{\Sigma'/E'} \models (\forall\{x, y : \mathit{Nat}\})\, x + y = y + x$, but $E' \not\models (\forall\{x, y : \mathit{Nat}\})\, x + y = y + x$.

Again, generalized theory morphisms compose and together with membership equational theories give rise to a category $\mathbf{GTh}_{\mathrm{MEL}}$.

The new feature of generalized signature morphisms, which is inherited by generalized theory morphisms, is that kinds and operators can be removed. This could have been "implemented" using the standard notion of theory morphism in the following alternative manner:

Proposition 2. *A generalized theory morphism* $H : T \longrightarrow T'$ *is the same thing as an ordinary theory morphism* $H : T \longrightarrow T' \oplus \mathit{ONE}$*, where* $\oplus$ *denotes coproduct of theories, and ONE is a theory with a single kind* $[\mathit{One}]$ *and sort One, a constant* $*$ *of that kind, and the equation* $(\forall\{x : [\mathit{One}]\})\, x = *$*.*

Proof (sketch). Leaving a kind or sort undefined in a generalized signature morphism corresponds respectively to mapping it to $[\mathit{One}]$ or *One* in $T' \oplus \mathit{ONE}$, while leaving an operator undefined corresponds to mapping it to the term $*$. □

Note that there is an equivalence of categories between the models of T' and those of $T' \oplus \mathit{ONE}$, because, even though we have introduced a new kind $[\mathit{One}]$, all its elements are collapsed by the equation $(\forall\{x : [\mathit{One}]\})\, x = *$ to the constant $*$ and can play no distinguished role.

Example. A special case of generalized theory morphisms are the projection functions from n-tuples to $(n-k)$-tuples. Consider a theory *3-TUPLE* for triples with kinds *3-Tuple*, *Elt@x*, *Elt@y*, *Elt@z*, an operator $\langle \_, \_, \_ \rangle : \mathit{Elt@x}\ \mathit{Elt@y}\ \mathit{Elt@z} \longrightarrow \mathit{3\text{-}Tuple}$, projection operators p_1, p_2, and p_3, and the obvious equations. Similarly, the theory *2-TUPLE* has kinds *2-Tuple*, *Elt@x*, *Elt@z*, an operator $\langle \_, \_ \rangle : \mathit{Elt@x}\ \mathit{Elt@z} \longrightarrow \mathit{2\text{-}Tuple}$, corresponding projection operators p_1 and p_2, and the equations for pairing. Projecting from a triple to a pair by projecting out the second component can be represented by the generalized theory morphism $H : \mathit{3\text{-}TUPLE} \longrightarrow \mathit{2\text{-}TUPLE}$ mapping the kinds *Elt@x* and *Elt@z* to themselves, *3-Tuple* to *2-Tuple*, and the operator $\langle \_, \_, \_ \rangle$ to the term $\langle x_1 : \mathit{Elt@x}, x_3 : \mathit{Elt@z} \rangle$; the image of the kind *Elt@y* and the operator p_2 are left undefined.

4.2 Simulation Maps as Generalized Theory Morphisms

To be able to arrange rewrite theories specifying Kripke structures in a categorical way we need to consider a theory $\mathit{BOOL}_{\models}$ extending BOOL with two new kinds, *State* and *Prop*, and a new operator $\_ \models \_ : \mathit{State}\ \mathit{Prop} \longrightarrow \mathit{Bool}$.

We now have all the ingredients needed to define a category $\mathbf{SRWThHom}_{\models}$ in which stuttering maps are specified by theory interpretations. Objects in $\mathbf{SRWThHom}_{\models}$ are triples $(\mathscr{R},(\Sigma',E\cup D),J)$ specifying, respectively, the transition relation, the atomic propositions, and the kind of the states. More precisely:

1. $\mathscr{R} = (\Sigma,E,R)$ is a rewrite theory specifying the transition system.
2. $(\Sigma,E) \subseteq (\Sigma',E\cup D)$ is a protecting theory extension, containing and protecting also the theory *BOOL* of Booleans, that defines the atomic propositions satisfied by the states. We define $\Pi \subseteq \Sigma'$ as the subsignature of operators of coarity *Prop*.
3. $J : \mathit{BOOL}_{\models} \longrightarrow (\Sigma',E\cup D)$ is a membership equational theory morphism [17] that selects the distinguished kind of states $J(\mathit{State})$, and such that: (i) it is the identity when restricted to *BOOL*, (ii) $J(\mathit{Prop}) = \mathit{Prop}$, and (iii) $J(\_ \models \_ : \mathit{State}\ \mathit{Prop} \rightarrow \mathit{Bool}) = \_ \models \_ : J(\mathit{State})\ \mathit{Prop} \rightarrow \mathit{Bool}$.

Then, a morphism

$$H : (\mathscr{R}_1,(\Sigma_1',E_1\cup D_1),J_1) \longrightarrow (\mathscr{R}_2,(\Sigma_2',E_2\cup D_2),J_2)$$

in $\mathbf{SRWThHom}_{\models}$ is a *generalized signature morphism* $H : \Sigma_1 \cup \Pi_1 \longrightarrow \Sigma_2 \cup \Pi_2$ such that:

1. $H \circ J_1 = J_2$ (so that *BOOL* is preserved and states in $\mathscr{R}_1$ are mapped to states in $\mathscr{R}_2$).
2. $H : (\Sigma_1,E_1) \longrightarrow (\Sigma_2,E_2)$ is a generalized morphism of membership equational theories with initial semantics, so that we have a unique Σ_1-homomorphism
$$\eta^H : T_{\Sigma_1/E_1} \longrightarrow U_H(T_{\Sigma_2/E_2}) : [t] \mapsto [H(t)]\,.$$
3. (Preservation of transitions.) $\eta^H_{J_1(\mathit{State})} : \mathscr{T}(\mathscr{R}_1)_{J_1(\mathit{State})} \longrightarrow \mathscr{T}(\mathscr{R}_2)_{J_2(\mathit{State})}$, the component corresponding to the kind $J_1(\mathit{State})$ in η^H mapping $[t]$ to $[H(t)]$, is a stuttering map of transition systems.
4. (Preservation of predicates.) For each $t \in T_{\Sigma_1,J_1(\mathit{State})}$ and state predicate $p(u_1,\ldots,u_n)$:
$$E_2\cup D_2 \vdash H(t) \models H(p(u_1,\ldots,u_n)) = \mathit{true} \implies E_1\cup D_1 \vdash t \models p(u_1,\ldots,u_n) = \mathit{true}\,.$$

We can analogously construct a subcategory $\mathbf{SRWThHom}^{\mathrm{str}}_{\models}$ of strict maps. The definition is exactly the same except for item (4), where the implication must actually be an equivalence.

That H so constrained indeed gives rise to a map of Kripke structures is shown in Proposition 3 below. Let us define a functor $\mathscr{K} : \mathbf{SRWThHom}_{\models} \longrightarrow \mathbf{KSMap}$ as follows:

- for objects, $\mathscr{K}(\mathscr{R},(\Sigma',E\cup D),J)=\mathscr{K}(\mathscr{R},J(State))_{\Pi}$;
- for morphisms $H:(\mathscr{R}_1,(\Sigma'_1,E_1\cup D_1),J_1)\longrightarrow(\mathscr{R}_2,(\Sigma'_2,E_2\cup D_2),J_2)$, $\mathscr{K}(H)=(H|_{\Pi_1},\eta^{H}_{J_1(State)})$, where $H|_{\Pi_1}$ is the restriction of H to the state predicates Π_1.

Proposition 3. *With the above definitions,* $\mathscr{K}:\mathbf{SRWThHom}_{\models}\longrightarrow\mathbf{KSMap}$ *is a functor with restriction* $\mathscr{K}:\mathbf{SRWThHom}^{\mathrm{str}}_{\models}\longrightarrow\mathbf{KSMap}^{\mathrm{str}}$.

Proof. $\mathscr{K}$ is well-defined on objects, and it is immediate to see that it preserves identities and composition of morphisms; the only thing we need to check is that, for all H, $\mathscr{K}(H)$ is indeed a map of Kripke structures. Let then $H:(\mathscr{R}_1,(\Sigma'_1,E_1\cup D_1),J_1)\longrightarrow(\mathscr{R}_2,(\Sigma'_2,E_2\cup D_2),J_2)$ be a morphism in $\mathbf{SRWThHom}_{\models}$. By item (3) above, $\eta^{H}_{J_1(State)}:\mathscr{T}(\mathscr{R}_1)_{J_1(State)}\longrightarrow\mathscr{T}(\mathscr{R}_2)_{J_2(State)}$ is a stuttering map of transition systems. To show preservation of predicates, let $p(u_1,\ldots,u_n)\in L_{\mathscr{K}(\mathscr{R}_2,J_2(State))_{\Pi_2}|_{H|_{\Pi_1}}}([H(t)])$. By definition of the reduct of a Kripke structure, $\mathscr{K}(\mathscr{R}_2,J_2(State))_{\Pi_2},[H(t)]\models H(p(u_1,\ldots,u_n))$ which, by definition of $\mathscr{K}(\mathscr{R}_2,J_2(State))_{\Pi_2}$ and condition (4) in the definition of morphisms in $\mathbf{SRWThHom}_{\models}$, implies that $p(u_1,\ldots,u_n)\in L_{\mathscr{K}(\mathscr{R}_1,J_1(State))_{\Pi_1}}([t])$, as required. It is clear that if H belongs to $\mathbf{SRWThHom}^{\mathrm{str}}_{\models}$ the converse is also true and $\mathscr{K}(H)$ is a strict map. □

An important consequence of this result and Theorem 1 is the following:

Theorem 2. *Given a morphism* $H:(\mathscr{R}_1,(\Sigma'_1,E_1\cup D_1),J_1)\longrightarrow(\mathscr{R}_2,(\Sigma'_2,E_2\cup D_2),J_2)$ *in* $\mathbf{SRWThHom}_{\models}$ *or* $\mathbf{SRWThHom}^{\mathrm{str}}_{\models}$*, and a formula* φ *in* $\mathrm{ACTL}^*\backslash\{\neg,\mathbf{X}\}(\Pi_1)$ *or* $\mathrm{ACTL}^*\setminus\mathbf{X}(\Pi_1)$ *respectively, if* $H(\varphi)$ *holds in* $\mathscr{K}(\mathscr{R}_2,(\Sigma'_2,E_2\cup D_2),J_2)$ *then* φ *holds in* $\mathscr{K}(\mathscr{R}_1,(\Sigma'_1,E_1\cup D_1),J_1)$.

Similar constructions can be carried out when simulations are represented by means of equationally defined functions or rewrite relations (recall the introduction to this section), resulting in categories $\mathbf{SRWTh}_{\models}$ and $\mathbf{SRelRWTh}_{\models}$. Then, the lifting of Kripke structures to the framework of rewriting logic can be represented graphically with the following commutative diagram. In it, the horizontal arrows between categories associated to Kripke structures are inclusions, and those that map to categories associated to transition systems are the expected forgetful functors. ($\mathbf{SRWTh}$ is constructed analogously to $\mathbf{SRWThHom}_{\models}$, but taking only the transitions into consideration.)

$$\begin{tikzcd}
\mathbf{SRWThHom}_{\models} \arrow[r] \arrow[d, "\mathscr{K}"] & \mathbf{SRWTh}_{\models} \arrow[r] \arrow[d, "\mathscr{K}"] & \mathbf{SRelRWTh}_{\models} \arrow[r] \arrow[d, "\mathscr{K}"] & \mathbf{SRWTh} \arrow[d, "\mathscr{T}"] \\
\mathbf{KSMap} \arrow[r] & \mathbf{KSMap} \arrow[r] & \mathbf{KSSim} \arrow[r] & \mathbf{STSys}
\end{tikzcd}$$

5 Applications

5.1 Predicate Abstraction

Simulations are useful to define abstractions that allow studying the properties of a complex system using a simpler one. A particular instance of the methodology of abstraction is *predicate abstraction* [13, 9]. Under this approach, the abstract domain is a Boolean algebra over a set of assertions and the abstraction function, typically as part of a Galois connection, is symbolically constructed as the conjunction of all expressions satisfying a certain condition, which is typically discharged using theorem proving. We now show how predicate abstractions can be understood as an instance of our notion of theoroidal map.

Let us first focus on the transition relation. Given a computational system, a set $\phi_1,\ldots,\phi_n$ of predicates over the states determines an abstraction function mapping a state S to the Boolean tuple $\langle\phi_1(S),\ldots,\phi_n(S)\rangle$. Let us assume that the transitions of the system are specified by a rewrite theory $\mathscr{R}=(\Sigma,E,R)$ whose kind of states is *State*. Then, if $\mathscr{R}$ is *State*-encapsulated with constructor $st : k_1\ldots k_m \longrightarrow State$ (that is, among all operators in Σ the kind *State* only appears in the operator *st*, and only as its coarity), the above predicate abstraction can be represented in rewriting logic by means of a rewrite theory $\mathscr{R}_A=(\Sigma_A,E_A,R_A)$ where:

- Σ_A contains Σ and the signature of *BOOL*, together with a new kind *BState*, a new operator $bst : Bool^n \longrightarrow BState$ and, for each predicate ϕ_i, $1\leq i\leq n$, an operator $p_i : State \longrightarrow Bool$ to represent it. We then have a signature morphism $H:\Sigma\longrightarrow\Sigma_A$ that maps the kind *State* to *BState*, the constructor *st* to the term

$$bst(p_1(st(x_1,\ldots,x_m)),\ldots,p_n(st(x_1,\ldots,x_m))),$$

  and is the identity everywhere else.
- E_A contains $H(E)$ and the equations in *BOOL*, together with equations for $p_1,\ldots,p_n$ specifying the predicates $\phi_1,\ldots,\phi_n$.
- $R_A = H(R)$.

By construction, then, $H:(\Sigma,E)\longrightarrow(\Sigma_A,E_A)$ is a theory morphism such that $t\rightarrow^1_{\mathscr{R},State} t'$ implies $H(t)\rightarrow^1_{\mathscr{R}_A,BState} H(t')$, thus preserving the transition relation.
We can now turn our attention to the preservation of properties. Graphically, the relationship between the different theories involved is depicted in the following diagram,

$$\begin{tikzcd}
(\Sigma,E) \arrow[r, hook] \arrow[d, "H"'] & (\Sigma',E\cup D) \arrow[d] \\
(\Sigma_A,E_A) \arrow[r, hook] & (\Sigma'_A,E_A\cup D_A)
\end{tikzcd}$$

where $(\Sigma',E\cup D)$ is the equational theory specifying the properties of the given system, and $(\Sigma'_A,E_A\cup D_A)$ is the theory we have to associate to $\mathscr{R}_A$ defining its atomic propositions.

The syntax for the state predicates q (that we assume are constants) in the original system is given in a subsignature Π of Σ'. It is usually the case that for each of these q one of the predicates ϕ_i in the basis defining the abstraction has the meaning “the state S satisfies q.” Let $q_1,\ldots,q_k$ be the state predicates in Π. We assume $k\leq n$, and that each q_j, $1\leq j\leq k$, corresponds to the predicate ϕ_j in the basis of the abstraction (but in general we may have $n>k$, with predicates $\phi_{k+1},\ldots,\phi_n$ not having a counterpart in Π). That is, for a ϕ_j with a corresponding q_j in Π, its specification in E_A through $p_j(S)$ is thus essentially the same (modulo renaming) as that of $S \models q_j$ in D, so that $E \cup D \vdash (S \models q_j) = \mathit{true} \iff E_A \vdash p_j(S) = \mathit{true}$. Then, for the abstraction we use the same set of state predicates Π and they are specified in a theory extension $(\Sigma_A, E_A) \subseteq (\Sigma'_A, E_A \cup D_A)$, with $\Sigma'_A = \Sigma_A \cup \Sigma'$ and D_A containing, for each q_j in Π with associated ϕ_j, the equation

$$(\forall\{x_1,\ldots,x_n\})\,(\mathit{bst}(x_1,\ldots,x_j,\ldots,x_n) \models q_j) = x_j\,.$$

Let us extend H to $\Sigma \cup \Pi$ by mapping each state predicate to itself. Thus, for all ground terms t of kind *State* and state predicates q_j, if $E_A \cup D_A \vdash (H(t) \models q_j) = \mathit{true}$ then, by the equation defining q_j in $E_A \cup D_A$ and since $H(t) = \mathit{bst}(p_1(t),\ldots,p_n(t))$, we have $E_A \cup D_A \vdash p_j(t) = \mathit{true}$ and even $E_A \vdash p_j(t) = \mathit{true}$ because p_j is completely specified in E_A. And hence, due to the relation between the equations defining $p_j(S)$ and $S \models q_j$, $E \cup D \vdash (t \models q_j) = \mathit{true}$ holds and preservation of predicates is guaranteed.

Finally, we can put all the pieces together and summarize the previous discussion as follows.

Theorem 3. *Let a concurrent system be specified as an object $(\mathscr{R}, (\Sigma', E \cup D), J)$ of* **SRWThHom**$_{\models}$*, where $\mathscr{R}$ is $J(\mathit{State})$-encapsulated, and let $\phi_1,\ldots,\phi_n$ be a set of predicates over the kind $J(\mathit{State})$, with each state predicate $q_j \in \Pi$ (we assume that all such q_j are constants) corresponding to a ϕ_j, $1 \le j \le k$. The result of applying predicate abstraction is the system given by $(\mathscr{R}_A, (\Sigma'_A, E_A \cup D_A), J_A)$, where $(\Sigma'_A, E_A \cup D_A)$ and $\mathscr{R}_A$ are defined as explained above, and where $J_A(\mathit{State}) = \mathit{BState}$. Then, with these definitions, $H : (\mathscr{R}, (\Sigma', E \cup D), J) \longrightarrow (\mathscr{R}_A, (\Sigma'_A, E_A \cup D_A), J_A)$ is an arrow in* **SRWThHom**$_{\models}$*, where H is the signature morphism $\Sigma \cup \Pi \longrightarrow \Sigma'_A \cup \Pi$.*

Let us illustrate these ideas by outlining how they apply to the bakery protocol. This is an infinite state protocol that achieves mutual exclusion between processes by dispensing a number to each process and serving them in sequential order according to the number they hold. For the case of two processes, the transitions can be specified in rewriting logic by a theory $\mathscr{R} = (\Sigma, E, R)$ such that:

- (Σ, E) contains declarations and equations specifying the natural numbers; in particular, the "equal to" (==) and "less than" (<) predicates are specified.
- States are constructed by an operator `st : Mode Nat Mode Nat -> State`. The first two components describe the status of the first process (the mode it is currently in, which can be `sleep`, `wait`, or `crit`, and its priority as given by the number according to which it will be served), and the last two components the status of the second process.
- R consists of eight rewrite rules, four for each process, describing all possible transitions. Among them, for example,

  ```
  rl st(M, X, sleep, Y) => st(M, X, wait, s(X)) .
  ```

  to represent that the second process can "awake" and move to `wait` mode, and

  ```
  crl st(M, X, wait, Y) => st(M, X, crit, Y) if Y < X .
  ```

  allowing the second process to move to the critical section if its counter is less than that of the first one.

The properties are defined in a theory extension $(\Sigma, E) \subseteq (\Sigma', E \cup D)$ that simply adds four constants `1wait`, `1crit`, `2wait`, and `2crit` to Σ to characterize when the first and second processes are in `wait` or `crit` mode, together with the obvious equations:

```
eq (st(wait, X, N, Y) |= 1wait) = true .
eq (st(sleep, X, N, Y) |= 1wait) = false .
eq (st(crit, X, N, Y) |= 1wait) = false .
...
```

For this protocol, we might be interested in verifying the following safety property: $\mathbf{AG}\neg(\texttt{1crit} \wedge \texttt{2crit})$.

We will use the following set of seven predicates to define the predicate abstraction:

$\phi_1(\texttt{st(M, X, N, Y)}) \iff \texttt{M == wait}$ $\quad\phi_5(\texttt{st(M, X, N, Y)}) \iff \texttt{X == 0}$
$\phi_2(\texttt{st(M, X, N, Y)}) \iff \texttt{M == crit}$ $\quad\phi_6(\texttt{st(M, X, N, Y)}) \iff \texttt{Y == 0}$
$\phi_3(\texttt{st(M, X, N, Y)}) \iff \texttt{N == wait}$ $\quad\phi_7(\texttt{st(M, X, N, Y)}) \iff \texttt{X < Y}$
$\phi_4(\texttt{st(M, X, N, Y)}) \iff \texttt{N == crit}$

Intuitively, we only care whether the processes are in `wait` or `crit` mode, whether their counters are equal to zero, and which counter is greater.

Note that the state predicates in the signature correspond to predicates 1–4. In terms of the notation used above, q_1 would be `1wait` and it would be associated to ϕ_1, q_2 would be `1crit` and would be associated to ϕ_2, and q_3 and q_4 would be `2wait` and `2crit`, associated to ϕ_3 and ϕ_4. Now, the abstract rewrite theory $\mathscr{R}_A = (\Sigma_A, E_A, R_A)$ is constructed by adding to $\mathscr{R}$:

– Operators `p1 : State -> Bool`,...,`p7 : State -> Bool`, together with a new kind `BState` and the constructor for abstract states

  ```
  op bst : Bool Bool Bool Bool Bool Bool Bool -> BState .
  ```

  This determines the signature morphism H, that maps the constructor operator `st` to the term

  ```
  bst(p1(st(M, X, N, Y)),...,p7(st(M, X, N, Y)))
  ```

– Equations associated to pi specifying ϕ_i for $i = 1, \ldots, 7$. Since predicates $\phi_1, \ldots, \phi_4$ correspond to the atomic propositions, their defining equations are "the same":

  ```
  eq p1(st(wait, X, N, Y)) = true .
  eq p1(st(sleep, X, N, Y)) = false .
  eq p1(st(crit, X, N, Y)) = false .
  eq p2(st(wait, X, N, Y)) = false .
  eq p2(st(sleep, X, N, Y)) = false .
  eq p2(st(crit, X, N, Y)) = true .
  ...
  ```

  The three remaining equations are also immediate:

  ```
  eq p5(st(M, X, N, Y)) = (X == 0) .
  eq p6(st(M, X, N, Y)) = (Y == 0) .
  eq p7(st(M, X, N, Y)) = (Y < X) .
  ```

– The translation of the rules in R by the signature morphism H. In particular, the two rules introduced before become:

```
rl bst(p1(st(M, X, sleep, Y)), ..., p7(st(M, X, sleep, Y))) =>
   bst(p1(st(M, X, wait, Y)), ..., p7(st(M, X, wait, s(X)))) .
crl bst(p1(st(M, X, wait, Y)), ..., p7(st(M, X, wait, Y))) =>
    bst(p1(st(M, X, crit, Y)), ..., p7(st(M, X, crit, Y)))
    if Y < X .
```

Finally, we have to write the equations in D_A defining the atomic propositions in the abstract model, which is straightforward.

```
eq (bst(B1, B2, B3, B4, B5, B6, B7) |= 1wait) = B1 .
eq (bst(B1, B2, B3, B4, B5, B6, B7) |= 1crit) = B2 .
eq (bst(B1, B2, B3, B4, B5, B6, B7) |= 2wait) = B3 .
eq (bst(B1, B2, B3, B4, B5, B6, B7) |= 2crit) = B4 .
```

By construction, this model is a predicate abstraction with respect to the basis $\phi_1, \ldots, \phi_7$ of the bakery protocol, in which the desired property can be model checked.

It is worth pointing out that this algebraic method of defining predicate abstractions cannot be expressed within the framework of [19], because the specification of the predicates ϕ_i requires, in general, to introduce auxiliary operators and thus a different signature $\Sigma_A \neq \Sigma$. Also, the resulting rewrite theory *is not executable* in general. This means that it cannot be directly used in a tool like the Maude model checker [10]. Predicate abstraction can be considered as a particular instance of our framework of algebraic simulations from a conceptual or foundational point of view, which is still quite useful because it provides a justification for the method within our framework. Current approaches to predicate abstraction do not work directly with the minimal transition relation (described in our account by $\mathscr{R}_A$). Instead, they compute a safe *approximation* of $\mathscr{R}_A$ by discharging some proof obligations. We are at present developing methods to compute such approximation within our framework using Maude's inductive theorem prover (ITP) [6] as the deductive engine to discharge such proof obligations.

5.2 A Fairness Example

We illustrate the use of theoroidal (bi)simulation maps to reason about fairness. The treatment can be made for very general classes of rewrite theories, and for quite flexible notions of fairness [18]. Here, we limit ourselves to illustrating some of the key ideas, including the use of theoroidal maps, by means of a simple communication protocol example. Note also that the same idea can be used for the representation and study of labeled transition systems in rewriting logic.

Consider a system consisting of a sender, a channel, and a receiver. The goal is to send a multiset of numbers (in arbitrary order) from the sender to the receiver through the channel. The channel can at any time contain several of these numbers. Besides the normal send and receive actions, the channel may stall an arbitrary number of times in sending some data. We can model the states of such a system by means of the signature

$$\begin{aligned} snd, ch, rcv &: Nat \longrightarrow Conf \\ null &: \longrightarrow Conf \\ \_\_ &: Conf\ Conf \longrightarrow Conf \end{aligned}$$

where the operator $\_\_$ (juxtaposition notation) denotes multiset union and satisfies the equations of associativity and commutativity, and has *null* as its identity element. For example, the term

$$snd(7)\,snd(3)\,snd(7)\,ch(2)ch(3)\,rcv(1)\,rcv(9)$$

describes a state in which 3 and two copies of 7 have not yet been sent, 2 and another copy of 3 are in the channel, and 1 and 9 have been received. The behavior of the system is specified by the following three rewrite rules:

$$\begin{gathered} send : snd(n) \longrightarrow ch(n) \\ stall : ch(n) \longrightarrow ch(n) \\ receive : ch(n) \longrightarrow rcv(n) \end{gathered}$$

where n is a variable of sort *Nat*. Is this system terminating? Not without extra assumptions, since the *stall* rule could be applied forever. To make it terminating it is enough to assume the following "weak fairness" property about the *receive* rule, described by the formula

$$wf\text{-}receive = \mathbf{FG}\,enabled\text{-}receive \rightarrow \mathbf{GF}\,taken\text{-}receive\,;$$

that is, if eventually the *receive* rule becomes continuously enabled in a path, then it is taken infinitely often. Specifying the *enabled-receive* predicate equationally is quite easy (we just need to have some value in the channel) but the specification of the *taken-receive* predicate is more elusive. For example, does the *taken-receive* predicate hold of the state described above? We don't know; maybe the last action was receiving the value 1, in which case it would hold, but it could instead have been stalling on 3, or sending 2, and then it wouldn't. Here is where a theory transformation corresponding to a theoroidal map, and allowing us to define a bisimilar system where the *taken-receive* predicate can be defined, comes in. The new theory extends the above signature with the following new sorts and operators:

$$\begin{gathered} send, stall, receive, * : \longrightarrow Label \\ \{\_ \mid \_\} : Conf\ \ Label \longrightarrow State \end{gathered}$$

that is, a state now consists of a configuration-label pair, indicating the last rule that was applied. Since initially no rule has been applied, we add the label $*$ for all initial states. The rules of the transformed theory are now:

$$\begin{gathered} send : \{conf\ snd(n) \mid l\} \longrightarrow \{conf\ ch(n) \mid send\} \\ stall : \{conf\ ch(n) \mid l\} \longrightarrow \{conf\ ch(n) \mid stall\} \\ receive : \{conf\ ch(n) \mid l\} \longrightarrow \{conf\ rcv(n) \mid receive\} \end{gathered}$$

where *conf* is a variable of sort *Conf*, and l a variable of sort *Label*. We can then define the predicates *enabled-send*, *enabled-receive*, and *taken-receive* by the equations

$$\begin{gathered} (\{conf\ snd(n) \mid l\} \models enabled\text{-}send) = true \\ (\{conf\ ch(n) \mid l\} \models enabled\text{-}receive) = true \\ (\{conf \mid receive\} \models taken\text{-}receive) = true \end{gathered}$$

Then the fair termination property can be defined by the following formula, which indeed holds in the Kripke structure associated to this transformed theory for any initial state:

$$\mathbf{A}(\textit{wf-receive} \rightarrow \mathbf{F}(\neg\textit{enabled-send} \wedge \neg\textit{enabled-receive}))\,.$$

Let $(\Sigma_{Comm}, E_{Comm})$ denote the underlying equational theory of our original rewrite theory, and let $(\Sigma_{LComm}, E_{Comm})$ denote that of the transformed theory (it has the same equations E_{Comm}). We can define a generalized theory morphism

$$H : (\Sigma_{LComm}, E_{Comm}) \longrightarrow (\Sigma_{Comm}, E_{Comm})$$

as follows. The sorts, implicit kinds, and operators in Σ_{Comm} are mapped identically to themselves; the sort *State* is mapped to *Conf*; and the sort *Label* is not mapped anywhere; the operator $\{\_ \mid \_\}$ is mapped to the variable *conf* of sort *Conf*; finally, the label constants are not mapped anywhere. Now, let Π_0 consist of the predicates *enabled-send* and *enabled-receive*, which in the original theory are defined by the equations

$$\begin{gathered} \textit{conf}\ \textit{snd}(n) \models \textit{enabled-send} = \textit{true} \\ \textit{conf}\ \textit{ch}(n) \models \textit{enabled-receive} = \textit{true}\,. \end{gathered}$$

Then, if *Comm* and *LComm* denote our rewrite theories, H induces a theoroidal *bisimulation* (strict) map of Kripke structures

$$H : \mathscr{K}(\textit{LComm}, [\textit{State}])_{\Pi_0} \longrightarrow \mathscr{K}(\textit{Comm}, [\textit{Conf}])_{\Pi_0}\,.$$

Furthermore, in the case of *LComm* we can extend Π_0 to Π by adding the *taken-receive* predicate, so that fair termination can be properly specified and verified.

6 Conclusions

We have argued that a categorical approach to the study of Kripke structures and their generalized notion of simulation is very natural, and have shown this by neatly organizing them in an institution. Among the many ways that these Kripke structures and simulations can be formally specified we have proposed rewriting logic, which has proved to be a very flexible framework for this task. Simulations come in several flavors in rewriting logic and here we have focused on theoroidal maps; we have shown how they can be organized together with rewrite theories in a category that reflects that for Kripke structures, and how they apply to two interesting examples. An open line of research consists in the study of proof methods and the development of tool support to prove simulations correct; some preliminary results are reported in [15].

References

1. M. Arrais and J. L. Fiadeiro. Unifying theories in different institutions. In M. Haveraaen, O. Owe, and O.-J Dahl, editors, *Recent Trends in Data Type Specification, COMPASS/ADT, Selected Papers*, volume 1130 of *LNCS*, pages 81–101. Springer-Verlag, 1996.

2. M. Barr and C. Wells. *Category Theory for Computing Science*. Centre de Recherches Mathématiques, third edition, 1999.
3. P. Borovanský, C. Kirchner, H. Kirchner, and P.-E. Moreau. ELAN from a rewriting logic point of view. *Theoretical Computer Science*, 285(2):155–185, 2002.
4. R. Bruni and J. Meseguer. Generalized rewrite theories. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Automata, Languages and Programming. ICALP 2003. Proceedings*, volume 2719 of *LNCS*, pages 252–266. Springer-Verlag, 2003.
5. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
6. M. Clavel. The ITP Tool. `http://geminis.sip.ucm.es/~clavel/itp`, 2004.
7. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2):187–243, 2002.
8. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. Maude manual (version 2.1). `http://maude.cs.uiuc.edu/manual/`, 2004.
9. M. A. Colón and T. E. Uribe. Generating finite-state abstractions of reactive systems using decision procedures. In A. J. Hu and M. Y. Vardi, editors, *Computer Aided Verification. CAV'98, Proceedings*, volume 1427 of *LNCS*, pages 293–304. Springer-Verlag, 1998.
10. S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker. In F. Gadducci and U. Montanari, editors, *Proceedings Fourth International Workshop on Rewriting Logic and its Applications, WRLA'02*, volume 71 of *ENTCS*. Elsevier, 2002.
11. K. Futatsugi and R. Diaconescu. *CafeOBJ Report*. World Scientific, AMAST Series, 1998.
12. J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.
13. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *Computer Aided Verification. CAV'97, Proceedings*, volume 1254 of *LNCS*, pages 72–83. Springer-Verlag, 1997.
14. N. Martí-Oliet and J. Meseguer. Rewriting logic: Roadmap and bibliography. *Theoretical Computer Science*, 285(2):121–154, 2002.
15. N. Martí-Oliet, J. Meseguer, and M. Palomino. Algebraic simulations. `http://maude.cs.uiuc.edu/papers/`, 2004.
16. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
17. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques, WADT'97, Selected Papers*, volume 1376 of *LNCS*, pages 18–61. Springer-Verlag, 1998.
18. J. Meseguer. Localized fairness: a rewriting semantics. Paper in preparation, 2004.
19. J. Meseguer, M. Palomino, and N. Martí-Oliet. Equational abstractions. In F. Baader, editor, *Automated Deduction - CADE-19. 19th International Conference on Automated Deduction, Proceedings*, volume 2741 of *LNCS*, pages 2–16. Springer-Verlag, 2003.
20. A. Tarlecki, R. M. Burstall, and J. A. Goguen. Some fundamental algebraic tools for the semantics of computation. Part 3: Indexed categories. *Theoretical Computer Science*, 91(2):239–264, 1991.