Electronic Communications of the EASST
Volume X (2011)

Proceedings of the
Tenth International Workshop on
Graph Transformation and
Visual Modeling Techniques
(GTVMT 2011)

# Towards a Maude Tool for Model Checking Temporal Graph Properties

Alberto Lluch Lafuente, Andrea Vandin

14 pages

Guest Editors: Fabio Gadducci, Leonardo Mariani
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

# Towards a Maude Tool for Model Checking Temporal Graph Properties[*]

**Alberto Lluch Lafuente, Andrea Vandin**

IMT Institute for Advanced Studies Lucca, Italy

**Abstract:** We present our prototypical tool for the verification of graph transformation systems. The major novelty of our tool is that it provides a model checker for temporal graph properties based on counterpart semantics for quantified $\mu$-calculi. Our tool can be considered as an instantiation of our approach to counterpart semantics which allows for a neat handling of creation, deletion and merging in systems with dynamic structure. Our implementation is based on the object-based machinery of Maude, which provides the basics to deal with attributed graphs. Graph transformation systems are specified with term rewrite rules. The model checker evaluates logical formulae of second-order modal $\mu$-calculus in the automatically generated Counterpart Model (a sort of unfolded graph transition system) of the graph transformation system under study. The result of evaluating a formula is a set of assignments for each state, associating node variables to actual nodes.

**Keywords:** Maude, Quantified $\mu$-calculi, counterpart semantics, verification, DPO

## 1 Introduction

Visual specification formalisms are nowadays used in almost the whole spectrum of software and hardware development activities. In the particular case of analysis and verification activities, visual specifications are complemented with appropriate property specification languages and tools for checking and verifying properties. A prominent example are graph transformation systems, temporal graph logics and the corresponding verification tools, which are used to reason about the possible transformations in a graph topology.

Recent approaches [BCKL07] propose variants of quantified $\mu$-calculi, resulting from a combination of the fix-point and modal operators of temporal logics with monadic second-order logic for graphs.These logics fit at the right level of abstraction for graph transformation systems: if state systems are graphs, and state components are thus graph items, one is not only interested in the topological structure of each reachable graph alone, but on its evolution as well.

Our own contribution to this trend of research was presented in [GLV10]. We introduced a novel semantics for quantified $\mu$-calculi. We considered a simple second-order syntax, and a notion of semantic model called *counterpart models* where states are algebras and the evolution relation is given by a family of partial homomorphisms. Instantiating our approach on graph transformation systems, states correspond to graphs and transitions correspond to the trace morphism. One of the main characteristics of our approach is that open formulae are interpreted over sets of pairs $(\sigma, w)$, for $w$ a state and $\sigma$ an assignment over $w$ (that is, a substitution associating

[*] This work has been supported by the European project ASCENS

formula variables to components of the state $w$). Our proposal avoids some limitations of other approaches, in particular in what regards the treatment of merging and name reuse. In addition, the resulting semantics is a streamlined and intuitively appealing one, yet it is general enough to cover most of the alternatives we are aware of.

In this paper we present our first step towards a tool support for our approach. In particular, we present first an implementation of graph rewriting as conditional rewrite rules on object multisets, which allows us to compositionally specify concurrent systems in an object-oriented fashion. Our system specifications are thus essentially graph transformation systems. Then we introduce a prototypical model checker that can be used to check quantified $\mu$-calculus formulae against system specifications. As far as we know, our tool is one of the few model checkers for a quantified $\mu$-calculus and one of the few ones based on counterpart semantics, allowing for a finer analysis of the evolution of individual components. Our work asses the feasibility of our approach, preparing the ground to build an efficient tool framework for verifying interesting properties of system specifications, possibly expressed in graph transformation style.

This paper is organised as follows. § 2 introduces a simple example. § 3 describes the basics of our approach, essentially an implementation of graph rewriting in Maude [CDE+07] to specify concurrent, multi-agent systems. § 4 summarizes our approach to counterpart semantics of quantified $\mu$-calculi. § 5 sketches the implementation details of the key functionalities of our prototype, which is put at work in § 6 with some examples. § 7 concludes the paper.

## 2 Running example

For a better illustration of our concerns, we consider the well known Stable Marriage Problem [GI89] as running example. We recall that the problem consists in finding a stable matching (i.e. a marriage) between $n$ men and $n$ women. Each individual has a preference ranking that orders all other individuals of opposite gender from the most preferred one to the least preferred one. A matching is *stable* if everyone is married, and there are not a man and a woman that would prefer to be married to each other, rather than with their current partners.

We have considered a simple distributed algorithm for solving the problem. Individuals are modeled as autonomous agents that communicate via asynchronous message passing in the form of tuple space communication, i.e. the algorithm abstracts away from the communication network, assuming that messages can be just delivered to the network indicating the receiver's id and can be picked up from the network with a sort of pattern matching. The distributed system contains three classes of entities: men, women and messages.

Initially, no marriage or message exist and all agents are single. In Figure 1 we give an intuitive graphical representation of an initial state with two men and two women. In section 3, we will better explain the format of the figure; intuitively, individuals and messages are represented as rounded boxes where the top frame contains the id and the sort (Male, Female or Message), and the bottom frame is reserved for attributes. Relations like "being married with" and "the n_th individual in my ranking", are graphically denoted with an edge going from the referring entity to the referred one labeled with the name of the relation. An edge with label $n$ from a male with id "m(i)" to a female with id "f(j)" express the fact that "f(j)" is in the $n\_th$ position of the ranking of "m(i)".

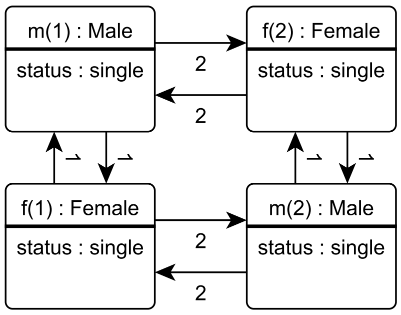

Figure 1: A graphical representation of an initial state with two agents of each kind.

A man that is single always sends a message containing the marriage proposal to the (next) preferred woman and waits for an answer. Women answer depending on their status and their ranking. A single woman always accepts a proposal. A married woman refuses proposals from men that are not preferred to their current partner. A married woman accepts the proposal from a man if she prefers him to her current partner, and sends a message to her partner notifying him that their marriage is broken.

When a man receives a divorce or a refuse notification, he becomes single and starts again sending a marriage proposal to the next most-preferred woman (eventually restarting from the most-preferred one). When a man receives an acceptance notification he gets married and remains idle, waiting for eventual divorce notifications or the end of the algorithm. The algorithm hence terminates when everyone is married, and terminates correctly if every marriage is stable.

An example of execution of such system is shown in Figure 2. The figure shows a four-step execution sequence, where states are displayed clock-wise starting from the top-left state: $s_0$,$s_1$,$s_2$,$s_3$. The evolution relation of the system, represented in the figure as gray fat arrows, goes from $s_0$ to $s_1$, from $s_1$ to $s_2$ and from $s_2$ to $s_3$. Intuitively, in state $s_0$ there are two single males "m(1)" and "m(2)", and two single females "f(1)" and "f(2)". In $s_1$ "m(1)" sends a marriage proposal to "f(1)" setting its own status to waiting. In $s_2$ "f(1)" accepts the proposal, sets its status to married, and sends the notification to m(1). Finally, in $s_3$ "m(1)" receives the notification, and the marriage is established.

In the rest of the paper we shall see how this algorithm can be modeled in Maude in a language based on graph-rewriting and how we can use Maude and our prototypical model checker to verify some properties of the algorithm. Indeed, some of the properties like the correctness for a given initial configuration can be verified with Maude's default tools, while others like individual mutual exclusion and response properties are directly verifiable in our tool only.

## 3 Graph rewriting with Maude

We have decided to rely our machinery on rewriting logic due to its well-developed theory based on the idea of computation as logical deduction, its expressiveness and generality witnessed by notable encodings of graph rewriting and programming languages, and its performant, easily ex-

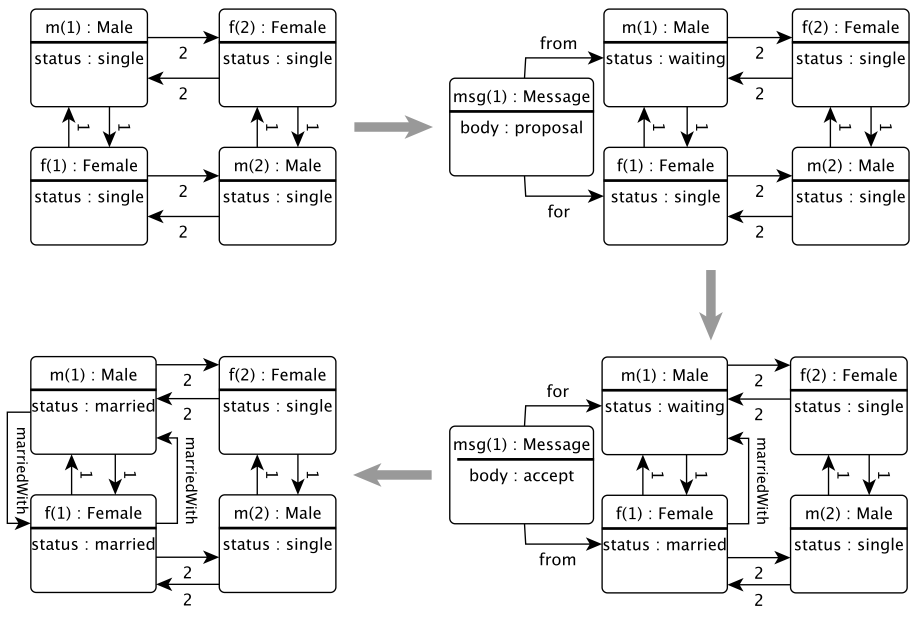

Figure 2: A graphical representation of a four step execution sequence.

tensible tool support. In particular, we will see that the configurations of our systems are terms of a particular signature of object configurations which correspond essentially to attributed graphs. The signature can be refined with concrete object classes, attribute types and well-formedness constraints that play the role of meta-modeling mechanisms such as type graphs. Last, the dynamics of a system is specified by rewrite theories, made in our case of rewrite rules in graph transformation style.

A *rewrite theory* $\mathscr{R}$ is a tuple $\langle \Sigma, E, R \rangle$ where $\Sigma$ is a signature, specifying the basic syntax (function symbols) and type machinery (sorts, kinds and subsorting) for terms, e.g. system configurations; $E$ is a set of (possibly conditional) equations, which induce equivalence classes of terms, and (possibly conditional) membership predicates, which refine the typing information; $R$ is a set of (possibly conditional) rules, e.g. graph rewrite rules.

The Maude framework [CDE+07] provides a language for describing such rewrite theories and a tool built upon a rewrite engine for executing and analysing them. In the rest of the paper we shall use Maude's syntax, introducing the syntactic ingredients as we use them.

The signature $\Sigma$ and the equations $E$ of a rewrite theory form a *membership equational theory* $\langle \Sigma, E \rangle$, whose initial algebra is $T_{\Sigma/E}$. Indeed, $T_{\Sigma/E}$ is the state space of a rewrite theory, i.e. states (e.g. graphs) are equivalence classes of $\Sigma$-terms modulo the least congruence induced by the axioms in $E$ (denoted by $[t]_E$ or $t$ for short). Operators are declared in Maude notation as `op f : TL -> T [a]` where `f` is the operator symbol (possibly with mixfix notation where argument placeholders are denoted with underscores), `TL` is a (possibly empty, blank separated) list of domain sorts, `T` is the sort of the co-domain, and `a` is a set of equational attributes (e.g. associativity, commutativity). We shall present a signature $\Sigma$ containing sorts and operators for describing models as collections of attributed, interrelated objects (i.e. attributed graphs).

Equations that cannot be declared as equational attributes must be treated as functions defined by a set of confluent and terminating (possibly conditional) equations of the form `ceq t = t′ if c`, where `t`, `t′` are Σ-terms, and `c` is an application condition. When the application condition is vacuous, the simpler syntax `eq t = t′` can be used. Roughly, an equational rule can be applied to a term `t′′` if we find a match for `t` at some place in `t′′` such that `c` holds (after the application of the substitution induced by the match). The effect is that of substituting the matched part with `t′` (after the application of the substitution induced by the match). One major advantage of Maude is that it includes tools for checking confluence, termination and completeness of equational logic specifications. The main equations of the theories we use allow us to treat object collections as multisets of objects, i.e. modulo associativity, commutativity, and identity (all treated as equational attributes), therefore axiomatising their graph-theoretic nature.

A membership predicate is of the form `cmb t : T if c`, where `t` is a Σ-term of some supersort `T′` of `T` and `c` is a predicate over `t` conditioning the membership statement. Roughly, a membership predicate states that if we are able to match a term `t′` with `t` such that `c` holds then `t′` has sort `T`. Membership predicates provide a subtyping mechanism that we can use, for instance, to check conformance wrt. to certain meta-model (e.g. typegraph).

Rewrite rules are of the form `crl t => t′ if c`, where `t`, `t′` are Σ-terms, and `c` is an application condition (a predicate on the terms involved in the rewrite, further rewrites whose result can be reused, membership predicates, etc.). When the application condition is vacuous, the simpler syntax `rl t => t′` can be used. Matching and rule application are similar to the case of equations with the main difference being that rules are not required to be confluent and terminating (as they represent possibly non-deterministic concurrent actions). Equational simplification has precedence over rule application in order to simulate rule application modulo equational equivalence. Rewrite rules can be used to program the behaviour of a system in a declarative way (e.g. in graph transformation style).

**Graphs as object collections** We summarize the previously mentioned algebra of object collections that is used to represent models as attributed graphs. In our setting a system configuration is a collection of attributed objects. Maude already provides a signature for this purpose, called object-based signature [CDE+07], which we tend to follow with slight modifications aimed to ease the presentation. Each object represents an entity (an individual component) and its properties. Technically, an object is defined by its identifier (of sort `Oid`), it's class (of sort `Cid`) and its attributes (of sort `AttSet`). Objects are built with an operation `< _ : _ | _ >` with functional type `Oid Cid AttSet -> Obj`. Object and Class identifiers will be defined by ad-hoc constructors. For instance in our running example we use the operation `m : Nat -> Oid` to use natural numbers to construct object identifiers for male individuals like `m(1)` or `m(2)`, and the constants `Male`, `Female` and `Message` of sort `Cid` to denote the classes of men, women and messages, respectively. The attributes of an object define its properties and relations to other objects. They are basically of two kinds: datatype attributes and relation attributes. Datatype attributes take the form `n: v`, where `n` is the attribute name and `v` is the attribute value. For instance, in our running example we shall consider an attribute `status` with domain in {`single,waiting,married`} (constants of sort `Status`), representing respectively whether a person is single, is waiting for a response or is married. Similarly, we will consider an attribute

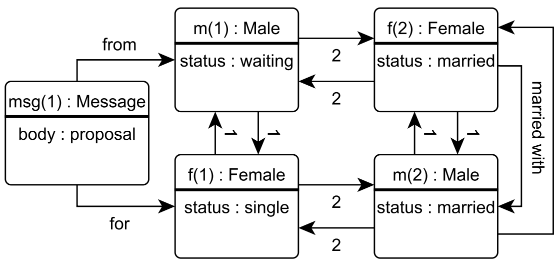

Figure 3: A graphical representation of a state.

body with domain in {proposal,divorce,accept,refuse} (sort MessageBody) for denoting respectively whether a message notifies a marriage proposal, a divorce or the acceptance or the refusal of a marriage proposal.

As an example of an attributed object, consider figures 1 and 3, whose format is reminiscent of the UML notation, with rounded boxes representing objects where the top frame contains the object identifier and its class, and the bottom frame is reserved for datatype attributes. Focusing only on the datatype kind of attributes, the man m(1) on the top-left of Figure 3 is denoted in Maude syntax with < m(1) : Male | status: waiting >.

In a configuration, objects are interrelated. Relations between objects can be represented in different ways. A very intuitive approach is to use a reference (a term of sort Oid) as value of an attribute. So if an object o1 has a relation R with object o2, then o1 will be equipped with an attribute R containing o2 in its value. Consider objects of class Message, they have a sender and a receiver. The message msg(1) of Figure 3 is a marriage proposal sent from the man m(1) for the woman f(1). The message is hence denoted as < msg(1) : Message | body: proposal , from: m(1) , for: f(1) >. Note that each relation is graphically denoted in Figure 3 with an arrow labelled with the name of the relation, which goes from the referring object to referred one. Even more complex relations can be graphically denoted in the same intuitive way. For example we represent the rankings of males and of females with an arrow labeled with the position of the referred man (woman) in the ranking of the referring woman (man). An arrow with label n from a woman f(i) to a man m(j) hence indicates that m(j) is in the n-th position of the ranking of f(i). A configuration can thus be thought of as a multi-sorted graph with attributes, where nodes correspond to objects, node attributes correspond to datatype attributes and labeled edges correspond to reference attributes. An object can be equipped with any number of attributes. Actually, the attributes of an object form a set built out of singleton attributes, the empty set (none) and union set (denoted with _, _).

Object configurations are essentially sets of objects. The sort for configurations is called Conf and its constructors are the empty configuration (none), singleton objects (as Obj is declared as subsort of Conf) and set union (denoted with juxtaposition). As an example the whole configuration of Figure 3 is denoted with

```
< msg(1) : Message | body: proposal , from: m(1) , for: f(1) >
< f(1) : Female | status: single , ranking: (m(1) |-> 1 , m(2) |-> 2) >
< f(2) : Female | status: married , ranking: (m(1) |-> 2 , m(2) |-> 1) ,
       marriedWith: m(2)  >
```

```
< m(1) : Male | status: waiting , ranking: (f(1) |-> 1 , f(2) |-> 2) >
< m(2) : Male | status: married , ranking: (f(1) |-> 2 , f(2) |-> 1) ,
      marriedWith: f(2) >
```

In order to distinguish a system configuration from the collection of objects that forms it, we wrap object collections together into a system with operation `<< _ >> :  Conf -> System`.

**Graph rewrite rules** To compositionally specify concurrent systems, we offer an object oriented language, based on an implementation of the double pushout approach (DPO) to graph rewriting: our systems can be hence seen as graph transformation systems specified by an initial state and a set of term rewrite rules given in DPO style. The main idea is that each rule has a left-hand side and a right-hand side pattern, each one composed by a set of objects (nodes) possibly interrelated by means of relation attributes (edges). In our tool we implement a two-level rule scheme: at the lowest level we have a set of *local rules* specific for every system, while at the top level we have a uniquely defined *global rule* that takes care of local rule application at the global level.

A local rule can be applied to a model whenever the left-hand side can be matched with part of the model, i.e. each object in the left-hand side is (injectively) identified with an object of the model respecting its relations. The global rule can be applied to a model whenever a local rule can be applied to part of the model and some additional application conditions hold, including the *no dangling edges* condition typical of graph transformation flavours like DPO. The choice of DPO is arbitrary and not a restriction, as we could also mimick other styles by changing the rule format, e.g. following SPO as in [BHM09].

Following our counterpart approach to the semantics of second-order $\mu$-calculus proposed in [GLV10], we do not implicitly identify elements of different systems, meaning that we do not have an implicit unique domain of objects, instead we enrich the rules with a partial morphism relating the objects matching the left-hand side pattern with the objects matching the right-hand side pattern. This morphism amounts to the trace morphism in graph rewriting and is used to intuitively express the preservation/renaming, deletion or fusion of objects, respectively if an object is mapped, it is not mapped, or more objects are mapped in the same one. An object appearing in the right-hand side pattern but not involved in the morphism is considered as a newly created one.

Considering our running example, the sending of marriage proposals is formalized by the local rule:

```
crl [makeProposal] :
  < idM : Male | status: single , ranking: (idF |-> nt , rankM) ,
        nextTry: nt , problemSize: size >
  < idF : Female | attSet1 >
=> {morphism}(
  < idM : Male | status: waiting , ranking: (idF |-> nt , rankM) ,
       nextTry: (s(nt) rem size) , problemSize: size >
  < idF : Female | attSet1 >
  < {new(0)} : Message |  body: proposal , from: idM , for: idF  > )
if morphism := (idM |-> idM , idF |-> idF) .
```

In this simple rule, a single man sends a marriage proposal to the “next most preferred woman”. The status and the `nextTry` counter of the involved man are hence updated, and a new object of

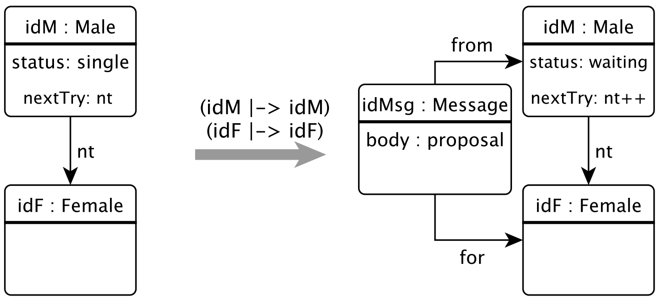

Figure 4: A graphical representation for the rule “makeProposal”.

class `Message` is created. The morphism also tells us that the man and the woman are preserved. The rule is intuitively graphically represented in Figure 4.

The global rule is instead defined as:

```
crl [global] :
  << conf remConf >> =>  {extMorphism} << conf3 conf4 >>
if conf => {morphism} conf3 /\
   noDanglingEdges(morphism , conf , remConf) /\
   extMorphism := extend(morphism,remConf) /\
   conf4 := applyToConfiguration(extMorphism , remConf) .
```

The global rule rewrites a system composed by the configurations `conf` and `remConf` into a system composed by the configurations `conf3` and `conf4`, correlating the two systems by the morphism `extMorphism` if 1) `conf` can be rewritten by a local rule in {`morphism`}`conf3`; 2) `morphism` does not delete objects of `conf` referred by objects in `remConf` (generating dangling edges); and 3) `conf4` is obtained applying `extMorphism` to `remConf`, where `extMorphism` is the extension of `morphism` with the identities in `remConf`. In other words, the global rule implements the pushout computation. Object creation is allowed and the consequent state explosion problem is partly mitigated by using name reuse, one of the more characterizing features approach, which allow us to deal with size bounded systems in case of systems with bounded resource allocation.

## 4 Counterpart Semantics for a Second-Order $\mu$-calculus

Many logics have been proposed to reason about the evolution of systems. In [GLV10] we introduced our own contribution with a novel semantics for a second-order $\mu$-calculus based on the Counterpart Theory proposed by Lewis and further developed in [Haz79]. Our proposal allows for a simple definition of the semantical universe by means of *Counterpart Models*, namely Kripke Models enriched with partial homomorphisms between connected worlds, called *counterpart relations*. Figure 5 denotes with dotted lines the counterpart relation between the states $s_1$ (top-right) and $s_2$ (bottom-right) of Figure 2. Intuitively, everything is preserved except for the message m(1) which is thus deleted and recreated evolving from state $s_1$ to state $s_2$. The two messages are not related: they share the same name, but represent two distinct components. It is important to notice that in the counterpart approach, the identifiers are local to the worlds they belong to. In different worlds, the same identifier may represent distinct elements.

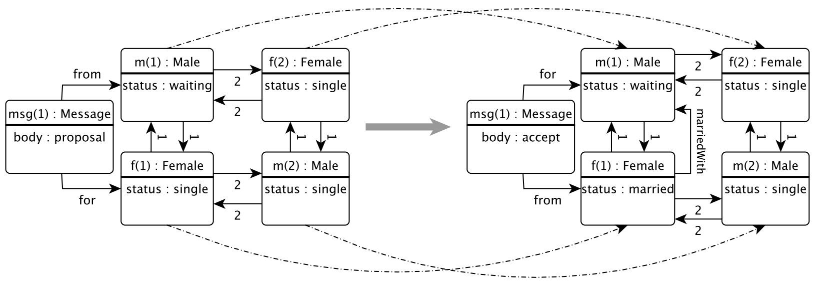

Figure 5: A graphical representation of the counterpart relation between two states.

Standard Kripke models identify elements through different worlds (trans-world identity), with implicitly defined identity morphisms, having as result a unique domain for the elements of the worlds. The presence of the unique domain, which is indeed just a technical solution, enforces restrictions of the evolution of states, making it difficult, or even forbidding to express merging, renaming, creation and deletion of elements. Enriching our models with the counterpart relations we avoid these limitations. For example, two elements are merged if they are mapped in the same element, while an element is a newly created one if it appears in the target state, but it is not involved in the counterpart function. In this manner counterpart models are well suited for modeling systems with dynamic structure. Moreover, since our semantics evaluates formulae with variables as sets of variable assignments for each world, instead of just worlds as in propositional logics and some non-propositional ones, it allows for a straightforward interpretation of fixed points and for their smooth integration with the evaluation of quantifiers, which are often dealt with by restricting the class of admissible models to those with no name reuse or merging. The resulting semantics is a streamlined and intuitively appealing one, yet it is general enough to cover most of the alternative proposals we are aware of . Now we briefly recall the syntax and semantics of our logic.

**Definition 1** (Formulae) Let $\Sigma$ be a signature (e.g. a signature for graphs), $\mathscr{Z}$ a set of fix-point variables, and $X$, $\mathscr{X}$ (denumerable) sets of first- and second-order variables typed over $\Sigma$ (e.g. node and node set variables). The set $\mathscr{F}_\Sigma$ of formulae of our logic is inductively generated by:

$$\psi ::= tt \mid \varepsilon : \tau \in_\tau \chi_\tau \mid \neg\psi \mid \psi \vee \psi \mid \exists x_\tau.\psi \mid \exists \chi_\tau.\psi \mid \Diamond\psi \mid Z \mid \mu Z.\psi$$

where $\varepsilon : \tau$ is a term over $\Sigma_X$ of type $\tau$, $\in_\tau$ is a family of membership predicates typed over $S_\Sigma$ indicating that (the evaluation of) a term with sort $\tau$ belongs to (the evaluation of) a second-order variable with the same sort $\tau$, and $\mu$ denotes the least fixed point operator.

We shall also derive the symbols $\wedge$ ,$\rightarrow$ , $\leftrightarrow$ , $\forall$, as well as well-known temporal operators like $\Box$ (all next steps), AG or AF (for all departing paths, always or eventually), and the greatest fix-point operator $\nu$ derived as $\nu Z.\psi \equiv \neg\mu Z.\neg\psi[^{\neg Z}/_Z]$, where $\psi[^{\neg Z}/_Z]$ stands for $\psi$ where all occurrences of $Z$ have been negated. Moreover, as it is standard, we restrict to *monotonic* formulae, i.e. such that each fix-point variable $Z$ occurs under the scope of an even number of negations. This is a sufficient condition for the fixed points to be well-defined. Note that the logic is simple, yet reasonably expressive. For instance, binary equivalence can also be defined as a derived operator, namely, $\varepsilon_1 : \tau =_\tau \varepsilon_2 : \tau$ is defined as $\forall \chi_\tau.\ (\varepsilon_1 : \tau \in_\tau \chi_\tau \leftrightarrow \varepsilon_2 : \tau \in_\tau \chi_\tau)$.

Our semantics does not evaluate naked formulae, but *formulae in context*, that is formulae enriched with informations about the free variables appearing in them. The context of a formula has two components: $\Gamma$ and $\Delta$ containing respectively first- and second-order variables. Reminiscent of the semantics of temporal formulae over sets of constraints introduced in [GHK00], the evaluation of a formula with context $[\Gamma;\Delta]$ consists in a set of pairs $(\sigma_w, w)$ where the domain of $\sigma_w$, a variable assignment for the world $w$, is defined exactly for the variables in $[\Gamma;\Delta]$. We indicate with $\Omega^{[\Gamma;\Delta]}$ the set of all the pairs of a model with assignments defined exactly for $[\Gamma;\Delta]$. The evaluation of a formula with empty context is hence just a set $\{(\lambda, w)\} \subseteq \Omega^{[\emptyset;\emptyset]}$, for $\lambda$ the empty variable assignment over the world $w$. Such an evaluation characterises a set of worlds, ensuring that our proposal properly extends the standard semantics of propositional modal logics.

The formulae of our logic are evaluated against *counterpart models*, which can be intuitively thought of as the graph transition system obtained by unfolding a graph transformation system. Intuitively, a counterpart model contains the informations graphically encoded both in Figure 2, and in Figure 5. Thus a counterpart model contains informations about the states of the system and their internal structure, and informations about the accessibility relation between the states, annotating explicitly the mappings between components of the distinct but connected states.

**Definition 2** (Semantics) Let $\psi[\Gamma;\Delta]$ be a formula-in-context (e.g stating some properties about the evolution of a graph), and $M$ be a counterpart model (e.g. the state transition graph obtained by unfolding a graph transformation system). The evaluation of $\psi[\Gamma;\Delta]$ in $M$ under the assignment $\rho : \mathscr{Z} \to 2^{\Omega^{[\Gamma;\Delta]}}$ is given by the function $[\![\cdot]\!]_\rho : \mathscr{F}^{[\Gamma;\Delta]} \to \Omega^{[\Gamma;\Delta]}$ defined as

$$\begin{array}{rcl}
[\![tt[\Gamma;\Delta]]\!]_\rho & = & \Omega^{[\Gamma;\Delta]} \\
[\![(\varepsilon : \tau \in_\tau \chi_\tau)[\Gamma;\Delta]]\!]_\rho & = & \{(\sigma, w) \in \Omega^{[\Gamma;\Delta]} \mid \sigma(\varepsilon) \in \sigma(\chi_\tau)\} \\
[\![\neg\psi[\Gamma;\Delta]]\!]_\rho & = & \Omega^{[\Gamma;\Delta]} \setminus [\![\psi[\Gamma;\Delta]]\!]_\rho \\
[\![\psi_1 \vee \psi_2[\Gamma;\Delta]]\!]_\rho & = & [\![\psi_1[\Gamma;\Delta]]\!]_\rho \cup [\![\psi_2[\Gamma;\Delta]]\!]_\rho \\
[\![\exists x_\tau.\ \psi[\Gamma;\Delta]]\!]_\rho & = & 2^{\downarrow x_\tau}([\![\psi[\Gamma, x_\tau;\Delta]]\!]_{(2^{\uparrow x}\circ\rho)}) \\
[\![\exists \chi_\tau.\ \psi[\Gamma;\Delta]]\!]_\rho & = & 2^{\downarrow \chi_\tau}([\![\psi[\Gamma;\Delta,\chi_\tau]]\!]_{(2^{\uparrow \chi}\circ\rho)}) \\
[\![\Diamond\psi[\Gamma;\Delta]]\!]_\rho & = & \{(\sigma, w) \in \Omega^{[\Gamma;\Delta]} \mid \exists(\sigma', w') \in [\![\psi[\Gamma;\Delta]]\!]_\rho\ .\ \sigma \overset{[\Gamma;\Delta]}{\rightsquigarrow} \sigma'\} \\
[\![Z[\Gamma;\Delta]]\!]_\rho & = & \rho(Z) \\
[\![\mu Z.\psi[\Gamma;\Delta]]\!]_\rho & = & \mathit{lfp}(\lambda Y.[\![\psi[\Gamma;\Delta]]\!]_\rho[{}^{Y}/_{Z}])
\end{array}$$

Note that in the evaluation of the membership predicate, $\sigma(\varepsilon)$ denotes the lifting of the substitution $\sigma$ to the set of terms over $\Sigma_X$. In the evaluation of the quantifiers, we make use of the functions $2^{\uparrow x}, 2^{\uparrow \chi}, 2^{\downarrow x}, 2^{\downarrow \chi}$ to respectively extend or restrict sets of pairs with the variable $x$ or $\chi$. Restricting a subset of $\Omega^{[\Gamma,x;\Delta]}$ respect to a variable $x$ we obtain a subset of $\Omega^{[\Gamma;\Delta]}$. Specularly, extending a subset of $\Omega^{[\Gamma;\Delta]}$ with a variable $x$ we obtain a subset of $\Omega^{[\Gamma,x;\Delta]}$. It is pivotal to require that the assignment $\rho$ for fix-point variables is extended to ensure a proper sorting of $\rho(Z)$, since it must now belong to the subsets of $\Omega^{[\Gamma,x;\Delta]}$ ($\Omega^{[\Gamma;\Delta,\chi]}$ in the second-order case). In the evaluation of the modal operator, the "renaming" of values across worlds is ensured by requiring that the assignments $\sigma$ and $\sigma'$ are in counterpart relation, meaning intuitively that $\sigma'$ respects $\sigma$ for the variables in $[\Gamma;\Delta]$. Hence all elements of $w$ assigned by $\sigma$ to the variables in $[\Gamma;\Delta]$ are mapped in $w'$ by the counterpart relation, respecting the operations on them. Thus, our semantics discards those worlds that are reachable but are not in counterpart with respect to the current context to avoid claims about non-existing elements (see [GLV10] for a detailed explanation).

## 5 Counterpart Model Generation and Model Checking

Our tool represents the first step towards a framework supporting our approach for the semantics of second-order $\mu$-calculi introduced in [GLV10]. We developed it aiming at assessing the feasibility of our approach providing a direct instantiation of it, leaving for future works concerns about efficiency and usability. Given that the formulae of our logic have to be checked against counterpart models, we first focused on their generation, and then we developed a model checker working on counterpart models.

**Counterpart Model generation** Counterpart models, as the well-known Kripke models, are defined by a triple `(W, d, RC)` where, `W` is a set of worlds, `d` is a function assigning a set of interrelated objects (a configuration) to each world in `W`, and `RC` is the accessibility relation between worlds. Respect to Kripke models, accessibility relations in counterpart models are equipped with partial homomorphisms, explicitly correlating elements of connected worlds. An entry of `RC` has the form of `w(i) =morphism=> w(j)` .

The procedure starts from a counterpart model containing only a world associated to an initial state and the empty accessibility relation. Then it keeps adding states and entries of the accessibility relation to the model up to completion of the state space. In particular, only two cases can arise after the generation of a state: the state is not already in the model, in which case a new world, the state and an accessibility relation entry are added to the model, or, in the second case, the state is already in the model, thus only the accessibility relation entry is added, if not already present. These two cases are captured by the following conditional rules:

```
crl (( W (d,(wSource |-> sSource)) RC)) =>
    (( (newWorld,W )                                 *** W
       (d,(wSource |-> sSource), (newWorld |-> sDest))   *** d
       (RC, wSource =morphism=> newWorld) ))         *** RC
if  sSource => {morphism}sDest /\
    systemNotInD(sDest , (d,(wSource |-> sSource))) .

crl ( (W ((wSource |-> sSource), (wDest |-> sDest) , d) RC) ) =>
    ( ( W                                            *** W
      ((wSource |-> sSource) , (wDest |-> sDest) , d)    *** d
      ( wSource = morphism => wDest , RC) ))         *** RC
if  sSource => {morphism}sDest /\
    notConnected(wSource, morphism, wDest, RC) .
```

It is worth to note that, the identification of syntactically identical states (equal graphs) is based on the reuse of object identifiers which allows us to obtain finite counterpart models in systems with bounded resource allocation. This happens, for instance in our example where the number of objects around is always bounded by a constant due to the message consumption and generation strategies. More powerful strategies based, e.g. on identifying symmetric states (isomorphic graphs) are under study.

Considering our running example, the counterpart model is built with the command `rew initializeCTModel(<< initSMP(n) >>)`, where `initSMP(n)` generates an initial state with `n` males (and females), and `initializeCTModel` generates the counterpart model containing only the initial state.

**Model Checking** Given a counterpart model `M` and an assignment for fix-point variables, our tool evaluates the semantics of a second order $\mu$-calculus formula as the set of pairs $(\sigma_w, w)$ satisfying it, where $w$ is a world of `M`, and $\sigma_w$ a variable assignment for $w$ defined exactly for the variables in the context of the formula. In doing so we first defined the operation `op valid` taking as arguments a formula in context, a pair, a fix-point variable assignment and a counterpart model. The operation reduces to true if the pair validates the formula in context, false otherwise. Finally, we evaluate the semantics of a formula in context with the operation `op [|_|]_,_` taking as arguments a formula in context, an initial state of the system (from which the counterpart model will be built), and an assignment for fix-point variables. Considering $[\Gamma;\Delta]$ as context of the formula, the operation generates all the pairs in the set $\Omega^{[\Gamma;\Delta]}$, and adds to the semantics of the formula only the ones for which `valid` is true.

## 6 Examples

The aim of this section is to illustrate the use of the tool to verify properties of the evolution of software systems, focusing on properties of individuals. For the rest of this section we fix an instance of our running example with `n = 2`, where all the people of the same gender have the same ranking.

**Individual response property.** In the algorithm sketched in section 2, people get married and divorced with the aim of finding particular marriages. An interesting property is the one stating that every time a male becomes single, he will later on become married. More formally, the property can be expressed as "for all male, whenever the male is single, it eventually becomes married". We can express the property with the formula $\psi$

```
forall xMale(0). AG((status(xMale(0)) = status: single)
                 -> (AF(status(xMale(0)) = status: married)))
```

Using the reduce command of Maude, we evaluate the semantics of $\psi$ with `reduce [|ψ|] <<initSMP(2)>> , empty`. As result we obtain a set of pairs $(\lambda, w(i))$ for all $w(i) \in W$, where $\lambda$ is the empty assignment. This tells us that the property holds in every state of the model.

**Individual mutual exclusion.** Other interesting properties regard the consistency of marriages. A meaningful example is "is it possible for two males to claim to be married with the same female?", expressed by the formula

```
not(xMale(0) = xMale(1)) and (marriedWith(xMale(0)) = marriedWith(xMale(1)))
```

Evaluating the property we find out that it holds in a world of the model, with the following assignment:

```
xMale(0) |-> < {m(2)} : Male | marriedWith: {f(1)}, ... >
xMale(1) |-> < {m(1)} : Male | marriedWith: {f(1)}, ... >
```

This can seem an erroneous scenario, but actually it happens because of the asynchronous and distributed fashion of the modelled algorithm: when a married woman accepts a new marriage proposal, she sends a divorce notification to the former partner and an accept notification to the new partner. In the case in which the accept notification is handled before the divorce one we have two males claiming to be married with the same woman. The consistency is restored at the next step, after the handling of the divorce notification. Different is the case in which we check the same property, but from the females perspective, just substituting `xMale` with `xFemale`. Evaluating the formula we obtain the empty set, meaning that it never happens in the model that two females claim to be married with the same male.

## 7 Conclusions and further works

Quantified modal logics have been studied in the realm of description logics (e.g. [FT03]), graph transformation (e.g. [BCKL07]), process algebras (e.g. [Cai04]) and model checking ([Ren06]) to cite a few. For a more comprehensive and detailed list we refer to [GLV10], where we also describe the differences with respect to our approach. Here we just mention that, as far as we know, graph transformation tools are not yet equipped with model checking capabilities for temporal logic other than propositional ones. Amongst them GROOVE[1] and AUGUR[2] seem the most promising one, since their authors have already produced interesting contributions to the theoretical foundations of model checking systems with dynamic structure using quantified temporal logics [Ren06, Ren03, DKR04, BCKL07].

The present paper introduces our prototypical tool to verify temporal graph properties, expressed in a quantified temporal logic. Our tool is based in the semantics for second-order $\mu$-calculus we introduced in [GLV10], which with respect to other approaches, allows for a simple definition of the semantical universe by means of counterpart models. The idea of associating to (open) formulae sets of assignments, instead of just worlds, allows for a straightforward interpretation of fixed points and for their smooth integration with the evaluation of quantifiers.

Our tool provides an instantiation of our approach, where formulae of our logic are checked against system specifications described in a graph-based dialect of Maude. In particular, we use a very popular Maude (sub)language for describing systems in a declarative, object-based style which essentially corresponds to graph rewriting. Such specifications can be analysed with Maude tools as usual, using for instance the critical pair analysis based confluence checker, the reachability analyzer or the propositional LTL model checker. Our implementation provides a finer model checker for formulae in a second-order $\mu$-calculus that allows to express more subtle properties like individual mutual exclusion or individual request-response.

In its current form, the model checker generates the entire counterpart model for a given specification and checks formulae on it. That is, our model checker does not yet verify properties on-the-fly, neither it does apply optimisation techniques based on symmetry or abstraction reduction. These issues are subject of current work as they could push our approach beyond its current bounded model checking form.

[1] http://groove.cs.utwente.nl/
[2] http://www.ti.inf.uni-due.de/research/augur/

## Bibliography

[BCKL07] P. Baldan, A. Corradini, B. König, A. Lluch Lafuente. A Temporal Graph Logic for Verification of Graph Transformation Systems. In Fiadeiro and Schobbens (eds.), *18th International Workshop on Recent Trends in Algebraic Development Techniques (WADT'06)*. LNCS 4409, pp. 1–20. Springer, 2007.

[BHM09] A. Boronat, R. Heckel, J. Meseguer. Rewriting Logic Semantics and Verification of Model Transformations. In *Proceedings of the International Conference on Fundamental Aspects of Software Engineering (FASE'09)*. LNCS 5503. Springer, 2009.

[Cai04] L. Caires. Behavioral and Spatial Observations in a Logic for the $\pi$-Calculus. In Walukiewicz (ed.), *7th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'04)*. LNCS 2987. Springer, 2004.

[CDE+07] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. L. Talcott. *All About Maude*. LNCS 4350. Springer, 2007.

[DKR04] D. Distefano, J.-P. Katoen, A. Rensink. Who is Pointing When to Whom? In al. (ed.), *32nd International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*. LNCS 3328. Springer, 2004.

[FT03] E. Franconi, D. Toman. Fixpoint Extensions of Temporal Description Logics. In Calvanese et al. (eds.), *16th International Workshop on Description Logics (DL'03)*. CEUR Workshop Proceedings 81. CEUR-WS.org, 2003.

[GHK00] F. Gadducci, R. Heckel, M. Koch. A Fully Abstract Model for Graph-Interpreted Temporal Logic. In Ehrig et al. (eds.), *6th International Workshop on Theory and Application of Graph Transformations (TAGT'98)*. LNCS 1764. Springer, 2000.

[GI89] D. Gusfield, R. W. Irving. *The stable marriage problem: structure and algorithms*. MIT Press, Cambridge, MA, USA, 1989.

[GLV10] F. Gadducci, A. Lluch Lafuente, A. Vandin. Counterpart Semantics for a Second-Order $\mu$-Calculus. In Ehrig et al. (eds.), *5th International Conference on Graph Transformation (ICGT'10)*. LNCS 6372, pp. 282–297. Springer, 2010.

[Haz79] A. Hazen. Counterpart-Theoretic Semantics for Modal Logic. *The Journal of Philosophy* 76(6):pp. 319–338, 1979.

[Ren03] A. Rensink. Towards Model Checking Graph Grammars. In Leuschel et al. (eds.), *3rd Workshop on Automated Verification of Critical Systems*. University of Southampton Technical Reports DSSE–TR–2003–2, pp. 150–160. 2003.

[Ren06] A. Rensink. Model Checking Quantified Computation Tree Logic. In Baier and Hermanns (eds.), *17th International Conference on Concurrency Theory (CONCUR'06)*. LNCS 4137, pp. 110–125. Springer, 2006.