Electronic Notes in Theoretical Computer Science 15 (1998)
URL: http://www.elsevier.nl/locate/entcs/volume15.html 21 pages

From Rewrite Theories to Temporal Logic Theories

Grit Denker [1]

Computer Science Laboratory
SRI International
Menlo Park, CA 94025
denker@csl.sri.com

Abstract

The work presented here aims at bridging the gap between executable specifications and formal verification. In this paper we combine two levels of description without changing the framework. The operational level of Maude/rewriting logic and the property-oriented level of temporal logics are combined. The combination is done by an embedding. We propose a distributed temporal logic as an extension of rewriting logic. Rewriting logic is primarily a logic *of* change in which the deduction directly corresponds to the computation. In contrast to that, temporal logic is a logic to talk *about* change in a global way. Especially, more complex system properties such as safety and liveness can be regarded in a temporal logic setting. In our approach we maintain the possibility of executing Maude specifications on the rewrite machine for validation purposes, and add the possibility of formally reasoning about Maude specifications in a temporal logic setting. The work presented focuses on object-oriented Maude specifications.

1 Introduction and Related Work

This paper proposes a distributed temporal logic as an extension of rewriting logic. Rewriting logic [Mes96b,Mes92] is primarily a logic *of* change in which the deduction directly corresponds to the change. In contrast to that, temporal logic is a logic to talk *about* change in a global way. Especially, more complex system properties such as safety and liveness can be regarded in a temporal logic setting. Both levels of description and analysis are useful in their own right; in fact, they complement each other. We therefore plan to use both logics in combination to prove properties about distributed systems. In this

[1] Work reported here was supported under HSPIII by German Federal and State Government through DAAD and partly by DARPA through Rome Laboratory Contract F30602-97-C-0312.

paper we provide the fundamentals of this goal by embedding rewriting logic into a temporal logic.

A variety of different modal or temporal logics can be chosen. We consider Distributed Temporal Logic (DTL) [ECSD98], a logic which is especially suited for capturing the nature of distributed systems. We intend to use temporal logic to express the dynamics of object-oriented specifications in Maude [Mes93] which is based on rewriting logic. We generalize DTL to DTL$^+$ since DTL assumes synchronous communication. In Maude systems, object communication may be synchronous, asynchronous, or a combination of both.

The main emphasis of DTL is to directly reflect the concurrent nature of a distributed system. For this purpose, the models of DTL$^+$ and DTL are concurrent labeled event structures which naturally express *causality, conflict,* and *concurrency*. Synchronization is modeled by shared events. The logics allow to formulate assertions about a system from the local viewpoints of the objects which belong to the system. We do not assume a global view on the system, but rather understand a system as a collection of concurrently existing and communicating objects and messages.

There exist other approaches to integrating a notion of time into rewriting logic or embedding rewriting logic into a framework capable of expressing temporal properties. Kosiuczenko and Wirsing [KW95] propose Timed Rewriting Logic to deal with time-sensitive systems. In their approach each rewriting step is labeled with a time stamp. Ölveczky and Meseguer [ÖM96] propose a semantic framework for modeling real-time and hybrid systems in rewriting logic. They do not extend rewriting logic but show how real-time systems can be formally specified in standard rewriting logic. The work presented here is not capable of dealing with real-time issues. Bridging the gap toward real time can be done by adapting ideas from real-time temporal logics such as Duration Calculus or Metric Temporal Logic (see [AH92] for an overview about logics of real time).

An approach closer to ours is the work done by Lechner [Lec97,Lec96]. She uses the μ-calculus for property-oriented descriptions of systems. Our work mainly differs from hers in the underlying model and in the way the use of temporal logics is integrated in the overall design process. First, Lechner defines Maude's semantics in the form of a labeled transition system. μ-formulas are interpreted over a (global) transition system. A state of the transition system represents a global state and, therefore, μ-formulas reflect global assertions. Second, we use temporal logic in a different way than Lechner does. Lechner proposes a three-level approach to specification: at the most abstract level, μ-formulas express properties of the intended system; at the intermediate level, formulas are blended with propositions on object states; and Maude is used at the concrete level. An appropriate notion of refinement is proposed to establish relationships between these levels. Our approach follows a different design process. We translate a given Maude specification into a temporal logic theory. There is no refinement involved, and specification and verification is done in the same temporal logic.

The work presented here is part of a more general approach toward bridging the gap between executable specifications and formal verification. In this paper we combine both these levels of description, the operational one of Maude and the property-oriented one of temporal logics, without changing the framework. We maintain the possibility of executing Maude specifications on the rewrite machine for validation purposes, and add the possibility of formally reasoning about Maude specifications in a temporal logic setting. The result of the mapping, the temporal logic theory, serves as a basis for further verification. Independent from the Maude specification, further desired (complex) properties of the system can be formulated. We focus here on the theory mapping and future work will deal with verification issues.

Section 2 gives an overview of the steps undertaken to transform an object-oriented Maude specification into a distributed temporal logic theory. Throughout the paper we use a simple example for illustrating the ideas. This example is introduced in Section 3. The syntax of the distributed temporal logic DTL^+ is presented in Section 4. Moreover, we define some special "logical macros" to deal with Maude specifications and illustrate the translation into a temporal theory by example. Section 5 discusses the model construction for a given rewrite theory. In Section 6 we conclude with some remarks on future work concerning temporal reasoning about Maude specifications.

2 Overview

The focus of this paper is on object-oriented Maude specifications. Our aim is to transform a Maude specification into a temporal logic theory. Our plan in achieving this is as illustrated in Figure 1.

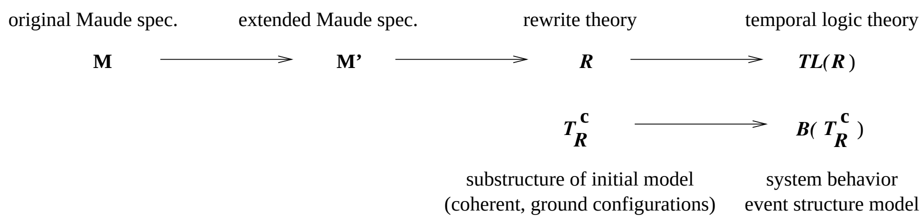

Fig. 1. Plan.

The first stage is to enrich a given object-oriented Maude specification with regard to the following three aspects. (1) Creation and deletion of objects and messages is reflected via attributes. Thus, we will be able to deal with object creation and deletion in our model as well as to distinguish between deadlocked versus deleted objects. (2) Messages are treated as objects with a very specific, restricted behavior. Thus, an appropriate identification mechanism for messages is proposed such that they can be uniquely identified over a system run, and messages have attributes. Configurations in which identification of objects and messages is unique are called coherent. (3) A rewrite rule to deal with initial configurations is established. This will be useful when we construct models. We want to make sure that only those system runs are considered which start from one of the specified initial configurations.

Given the extended specification we can transform it into a rewrite theory $\mathcal{R} = (\Sigma, E, L, R)$ with a signature Σ, equations E, labels L, and rewrite rules R. The rewrite theory is transformed into a temporal logic theory $\mathcal{TL}(\mathcal{R})$ by keeping the signature and equations and transforming rewrite rules into temporal formulas. Additional frame rules are defined.

The initial model $\mathcal{T}_{\mathcal{R}}(X)$ of $\mathcal{R}$ is a category whose objects are equivalence classes of terms and whose morphisms are equivalence classes of proof terms representing proofs in rewriting deduction, that is, concurrent $\mathcal{R}$-rewrites. We consider a substructure of this model. $\mathcal{T}^c_{\mathcal{R}}$ is the substructure of the initial model in which all proof terms start with a coherent, initial, ground configuration and transform between coherent ground configurations (details are given in Section 5.2). This substructure serves as a basis to construct an event structure model. More particularly, we construct a concurrent event structure model $\mathcal{B}(\mathcal{T}^c_{\mathcal{R}})$ over the proof terms in $\mathcal{T}^c_{\mathcal{R}}$. We then show that the constructed model is a model for the temporal logic theory $\mathcal{TL}(\mathcal{R})$.

3 A Simple Example: `Communicating Variables`

The emphasis of this work is on object-oriented specifications. Thus, we assume rewrite rules of an object-oriented specification, that is, labeled rules with a set of objects and messages on both sides of the rule. The general form required of rewrite rules in an object-oriented specification of Maude is

$$r : M_1 \ldots M_n \;\; O_1 \ldots O_m \longrightarrow O^{'}_{i_1} \ldots O^{'}_{i_k} \;\; Q_1 \ldots Q_p \;\; M^{'}_1 \ldots M^{'}_q \;\; \textit{if } C$$

where $k, p, q \geq 0$, the M's are message expressions, $i_1, \ldots, i_k$ are different numbers among the original $1, \ldots, m$, and C is the rule's condition. We leave out the possibility of class migration. In future work we will investigate how this can be incorporated into our framework.

We illustrate the object-oriented concepts of Maude and the main ideas about the transformation steps by means of an example: variables which communicate by message passing. A similar example can be found in [ECSD98].

3.1 Original Maude Specification

Let us consider a communication system which consists of two objects, a sender `s` and a receiver `r`. Both objects represent variables which can store a natural number (`val`), and have a set of natural numbers they can choose from to change the value of the attribute `val`. The sender stores the identity of its communication partner (`rec`). The sender object alternately may pick one number from its set and store it as the current value or forward the current value to its communication partner while setting the value to zero.

Independently from the receiver object's value, after receiving a message from the sender object the receiver value is set to zero.

```
omod CommunicatingVariables is

sorts Nat NatSet .
subsort Nat < NatSet .
ops 0 1 2 3: ->  Nat .
op _._: NatSet NatSet -> NatSet [comm assoc] .

class Variable | val: Nat, set: NatSet .
class Sender | rec: OId .
class Receiver .
subclass Sender Receiver < Variable .

msg to : OId Nat -> Msg .

var S R V : OId .  vars N M : Nat . var Set : NatSet .

crl [choose_and_change] :
 < V ; Variable | val: 0, set: N.Set > => < V ; Variable | val: N >
 if N =/= 0 .

crl [send] :
 < S ; Sender | val: N, rec: R > => < S ; Sender | val: 0 > to(R,N)
 if N =/= 0 .

crl [receive] :
 < R ; Receiver | val: N > to(R,M) => < R ; Receiver | val: 0 >
 if N =/= 0 .

op Initial : -> Configuration .
ops s r: OId .
eq  Initial = < s ; Sender | val: 0, set: 1.2.3, rec: r >
              < r ; Receiver | val: 0, set: 1.2.3 > .

endom
```

The conditions of the rules imply that a sender object can only alternately execute the rules `choose_and_change` and `send`. Analogously, the receiver object may only alternately receive a message or change its value.

3.2 Extended Maude Specification

Our goal is to extend an object-oriented Maude specification due to several requirements. The main emphasis is to treat objects and messages in the same way. Let us first motivate this goal before we go into details concerning the extension of specifications.

Unlike in other object-oriented approaches, messages are not associated with objects in Maude. Messages exist independently from objects, and as such they may be created and deleted independently from a specific object. As a consequence, a variety of communication principles is expressible in Maude. For instance, asynchronous and synchronous communication between two or more objects can be described by rewrite rules. Since the form of a rewrite rule does not restrict the way in which objects and messages may appear, messages cannot be handled as parts of objects. As a solution we have decided to treat messages and objects in the same way. As will become clear in Section 4, our logic provides principles to express a system as a set of communicating agents. In our setting, agents are either objects or messages. Therefore, we decided to treat messages as objects which behave in a very restricted way. A message may be created with a specific content and it may be deleted in the next step or it will remain in the system for the rest of its life. Thus, a message life cycle essentially consists of one or two states, the state after its creation and possibly one more state if it is destroyed. To treat messages like objects, we must be able to uniquely identify messages and to talk about the state of a message (i.e., existing vs. destroyed, the message's content).

For our approach we assume that Maude specifications are transformed in such a way that (1) messages are uniquely identifiable and have attributes, (2) creation and deletion of objects and messages is reflected in attributes, and (3) initial configurations are understood as special rewrite rules on a prenatal configuration. There are different ways to achieve these three issues. We will propose a specific solution which will be the basis to prove the soundness of the event structure construction to be presented in Section 5.2.

ad (1): The way in which messages are usually specified in Maude specifications does not guarantee their uniqueness over a possible system run. For instance, given a configuration of a specification, several messages with identical names and values may exist. To uniquely identify the occurrence of a message, that is, a message instance during the computation of the system, we must introduce further parameters. A solution proposed in [MT98] is to introduce an extra counter for every object and use this counter together with the object identifier as an identity for the message. Generally, it is sufficient to use the identity and counter of one object that is involved in the creation of a message. Thus, we assume a message identifier sort `MId`. At the end of this section we propose a solution for our running example.

ad (2): Dynamic creation and deletion of objects and messages require special treatment. In our model, objects and messages are modeled by (sets of) life cycles, that is, sequences of events. An infinite life cycle represents an object which persistently exists. A finite life cycle may represent either (a) a deadlocked object or message or (b) an object or message which temporarily existed in the system and has been destroyed. To distinguish between a deadlock state, that is, a state in which an object or a message exists but can no longer evolve, and a state which is the last one in a life cycle before the object or message has been destroyed, we introduce a boolean existence attribute which is abbreviated by $*$. If $* = \mathit{true}$ holds in the last state of a finite life cycle then the object or message exists and is deadlocked; if $* = \mathit{false}$ holds in the last state of a finite life cycle then this object or message is deleted. A rewrite rule of the original specification is modified in the following way. First, each object and message on the left-hand side of the rule is extended by enforcing $* = \mathit{true}$ to be valid. Second, each object or message deleted in a rule is copied to the right-hand side with $* = \mathit{false}$. In this way, objects or messages in a configuration are not deleted; rather, they are marked as destroyed. Third, each new object or message is created with the value $* = \mathit{true}$.

Moreover, for each message we assume a content attribute `cnt` to be defined. The content of a message consists of the message name and its parameter values. Thus, we can think of a message in the same way we understand objects. A message has an identity and it has two attributes, one for its existence and another to store its content. Therefore, we define a class of messages: `class Message | cnt: Msg, *: Bool`. We can represent messages in the same way we represent objects, that is, a message with identity `m` is represented as `< m; Message | cnt: v, *: b >`. A possible transformation for our running example is given at the end of this section.

ad (3): Our models are sets of possible system runs where each run starts in one of the defined initial configurations. Each proof term represents a finite behavior. We introduce a new rule `start` which is by default contained in every Maude specification. Applying this rule delivers an initial configuration: `rl [start]: PreNatal => C`, `C: IConf`. `IConf` is a subsort of the sort `Configuration`. `PreNatal` is defined as a special configuration: `op PreNatal : -> Configuration`. A declared initial configuration of a Maude specification can be used to instantiate the `start` rule. Let `op Initial : -> IConf` be an operation definition and let `[C <- Initial]` be a notation to express variable assignment. Then, we can derive the following instantiated rule `start([C <- Initial]): PreNatal => Initial`. A specification transformation satisfying all mentioned requirements may result in the following extended Maude specification.

```
omod CommunicatingVariables' is

sorts Nat NatSet .
subsort Nat < NatSet .
ops 0 1 2 3: -> Nat .
op _._: NatSet NatSet -> NatSet [comm assoc].

sorts IConf MId .
subsort IConf < Configuration .
class Variable | val: Nat, set: NatSet, *: Bool .
class Sender | rec: OId, counter: Nat .
```

```
class Receiver .
subclass Sender Receiver < Variable .

class Message | cnt: Msg, *: Bool .
op (_,_): OId Nat -> MId .
msg to: OId Nat -> Msg .

var S R V : OId .  vars N M L : Nat . var Set : NatSet .

crl [choose_and_change] :
 < V ; Variable | val: 0, set: N.Set, *: true >
 => < V ; Variable | val: N > if N =/= 0 .

crl [send] :
 < S ; Sender | val: N, rec: R, counter: M, *: true >
 => < S ; Sender | val: 0, counter: M+1 >
    < (S,M); Message | cnt: to(R,N), *: true > if N =/= 0 .

crl [receive] :
 < R ; Receiver | val: N, *: true >
 < (S,L); Message | cnt: to(R,M), *: true >
 => < R ; Receiver | val: 0 > < (S,L); Message | *: false >
 if N =/= 0 .

var C: IConf .
op PreNatal : -> Configuration .

rl [start]: PreNatal => C .

ops s r: OId .
op Initial : -> IConf .
eq Initial =
   < s ; Sender | val: 0, set: 1.2.3, rec: r, counter: 1, *: true >
   < r ; Receiver | val: 0, set: 1.2.3, *: true > .

endom
```

4 The Logic DTL$^+$

DTL$^+$ is a distributed temporal logic in the spirit of DTL and D_1 [ECSD98]. DTL and D_1 assume synchronous communication as the given communication principle. The main purpose of DTL$^+$ is twofold: (1) DTL$^+$ supports several types of communication among which one can find synchronous and asynchronous communication; (2) DTL$^+$ is designed in such a way that it can be used as a semantic basis for Maude specifications, and, in this way, extends rewriting logic by temporal operators. These goals are achieved by explicitly incorporating objects as well as messages in the logic and in the interpretation structures.

4.1 Syntax

Our logic is parameterized over rewrite theories. An extended Maude specification can be translated into a rewrite theory $\mathcal{R} = (\Sigma, E, L, R)$ where $\Sigma = (S, \Omega)$ is a signature with sorts S and operation symbols Ω. The "desugaring" process of transforming object-oriented modules to system modules was originally described in [Mes93]. The currently implemented version of flattening object-oriented modules to system modules is presented in [CDELM98,DM98]. For our purposes it is sufficient to point out which sorts and operations we assume to be at least in Σ. Thus, we will not go into the details of transforming object-oriented modules to system modules but rather mention those sorts and operations that we assume to be in Σ. Later we define DTL^+ as a logic which is parameterized over data terms over a signature which provides at least the necessary sorts and operations.

A configuration in Maude is a multiset of objects and messages. We assume a sort *Configuration* in Σ. To deal with initial configurations we introduce a sort $\mathit{IConf} < \mathit{Configuration}$. Each initial configuration is a term of sort *IConf*. Moreover, we assume a sort *CConf*, $\mathit{IConf} < \mathit{CConf} < \mathit{Configuration}$ of coherent configurations. In a coherent configuration, identifiers for messages and objects are unique. We come back to this notion in Section 5. Each class name `C` defines a data sort *CId* of identities of objects of that class. *CId* is a subsort of the sort of object identities *OId*. We assume a sort for attribute-value pairs *Att*. Thus, each attribute declaration `att:s` gives rise to a function $\mathit{att} = \_ : s \rightarrow \mathit{Att}$ where $\mathit{att} = v$ means that the attribute with identity *att* has value v. For each class we have a subsort $\mathit{CAtt} < \mathit{Att}$ of attributes of that class. $\mathit{att} = v$ is of sort *CAtt* only if `att:s` has been defined for class `C` and `v` is a value of corresponding sort. $\mathit{C1Att} < \mathit{C2Att}$ holds if and only if *C1* is a subclass of *C2*. We assume a specific sort of message identities *MId*. Two attributes are defined for messages. The content of the message is described with the help of the attribute-value pair $\mathit{cnt} = \_ : \mathit{Msg} \rightarrow \mathit{MAtt}$. A second attribute expresses whether a message exists or is deleted, that is , $* = \_ : \mathit{Bool} \rightarrow \mathit{MAtt}$. The attribute $*$ is also declared for each object class and give rise to a function $* = \_ : \mathit{Bool} \rightarrow \mathit{CAtt}$. This framework allows us to deal with objects and messages in the same way. The main difference between objects and messages is that messages have a much more restricted behavior than objects: a message with a specific content is created in the system and assigned a unique identifier. Either this message remains unchanged in the system or it is destroyed by another object which consumes the message. The frame rules describing the restricted behavior of messages are presented below. To summarize, the extended Maude specification determines a signature $\Sigma = (S, \Omega)$ with a set of sorts S. Given an S-indexed set $X = \{X_s\}_{s \in S}$ of variable symbols, the Σ-terms over X are denoted by $T_\Sigma(X)$.

For our running example the following is a sketch of part of the signature which can be derived from the extended Maude specification after the flattening process:

$$\begin{aligned}
S \supset \{ & \mathit{VariableId},\ \mathit{SenderId},\ \mathit{ReceiverId},\ \mathit{MId},\ \mathit{VariableAtt}, \\
& \mathit{SenderAtt},\ \mathit{ReceiverAtt}\}, \\
\Omega \supset \{ & \mathit{val} = _ \ : \mathit{Nat} \rightarrow \mathit{VariableAtt}, \\
& \mathit{set} = _ \ : \mathit{NatSet} \rightarrow \mathit{VariableAtt}, \\
& * = _ \ : \mathit{Bool} \rightarrow \mathit{VariableAtt}, \\
& \mathit{rec} = _ \ : \mathit{OId} \rightarrow \mathit{SenderAtt}, \\
& \mathit{counter} = _ \ : \mathit{Nat} \rightarrow \mathit{SenderAtt}, \\
& \mathit{cnt} = _ \ : \mathit{Msg} \rightarrow \mathit{MAtt}, \\
& * = _ \ : \mathit{Bool} \rightarrow \mathit{MAtt}, \\
& (_,_) : \mathit{OId}\ \ \mathit{Nat} \rightarrow \mathit{MId}, \\
& \mathit{to} : \mathit{OId}\ \ \mathit{Nat} \rightarrow \mathit{Msg}, \\
& s : \ \rightarrow \mathit{SenderId}, \\
& r : \ \rightarrow \mathit{ReceiverId}\}.
\end{aligned}$$

Translating the operation definition in the extended Maude specification would determine $s, r\ :\ \rightarrow OId$, but taking the initial configuration into account, we get the more specialized information as given above.

Definition 4.1 *Let $\Sigma = (S, \Omega)$ be a signature, let $t_1, t_2 \in T_\Sigma(X)_s (s \in S)$ be data terms, let i be an identity term, i.e., an object identity term of some class C ($i \in T_\Sigma(X)_{CId}$) or a message identity term ($i \in T_\Sigma(X)_{MId}$), let a_i be an attribute term for i, i.e., if i is a message identity term then $a_i \in T_\Sigma(X)_{MAtt}$ and if i is an object identity term of class C then $a_i \in T_\Sigma(X)_{CAtt}$. The syntax of* $\textsc{Dtl}^+$ *is defined as follows:*

$$\begin{aligned}
\textsc{Dtl}^+ \quad &::= \{\textsc{Dtl}^+{}_i\}_{i \in (\cup_C T_\Sigma(X)_{CId} \cup T_\Sigma(X)_{MId})}, \\
\textsc{Dtl}^+{}_i \quad &::= i\text{: } (\mathit{locTL}_i), \\
\mathit{locTL}_i \quad &::= t_1 =_s t_2 \mid a_i \mid \mathit{false} \mid (\exists x)\ \mathit{locTL}_i \mid \mathit{locTL}_i \Rightarrow \mathit{locTL}_i \\
&\qquad \mid \mathit{locTL}_i\ \mathsf{U}\ \mathit{locTL}_i \mid \mathit{locTL}_i\ \mathsf{S}\ \mathit{locTL}_i \mid \rhd\ \mathit{locTL}_i \mid \mathit{comL}, \\
\mathit{comL} \quad &::= \textsc{Dtl}^+{}_j \text{ for some identity term } j \neq i.
\end{aligned}$$

U and S are the *until* and *since* temporal operators. The $\rhd$ operator is used to express enabledness of transitions. $\rhd\varphi$ means that there exists a successor state in which φ holds. As usual, we introduce derived connectives such as $\neg, \mathit{true}, \vee, \mathsf{Y}$ (yesterday/previous) and X (next).

The main difference between $\textsc{Dtl}^+$ and its predecessors $\textsc{Dtl}$ and D_1 is that the underlying signature of $\textsc{Dtl}^+$ incorporates some specific sorts such as message identity and object identity. As a consequence, asynchronous message passing is naturally expressible in $\textsc{Dtl}^+$. In contrast to that the underlying communication principle in these logics is synchronous. Let us illustrate a way of expressing asynchronous message passing in $\textsc{Dtl}^+$. Assume φ is a formula about the state of object o, then $o : (\varphi \Rightarrow \mathsf{X}(\varphi' \wedge m : (* = \mathit{true} \wedge \mathit{cnt} = c)))$

can be interpreted as *if object o is in a state φ then it sends out message m with content c and changes its state to φ'*. The formula $o : ((\varphi \wedge m : (* = \mathit{true})) \Rightarrow \mathsf{X}(\varphi' \wedge m : (* = \mathit{false})))$ says that *a system configuration, with an object o in state φ and a message m, goes over to a configuration where m is deleted and o changes its state to φ'*. One may read this formula as *o consumes message m* since the only prerequisite to delete message m is the existence of object o.

In our approach we treat objects and messages in the same way. They both have attributes and behave over time. The main difference between objects and messages is that messages have a much more restricted behavior. Messages are usually less persistent than objects. A message is created with a specific content and it either remains unchanged in the system or it is deleted. In contrast to objects, a message may not change the value of its content attribute. Thus, one can understand messages as objects without methods with the exception of creating and deleting a message. Using DTL^+ we can formalize frame rules for messages and objects.

For objects as well as messages it is true that after deletion there exists no other state. The temporal operator $\mathsf{X}^?$ is defined as $\mathsf{X}^?\varphi = \neg(\mathsf{X}(\neg\varphi))$, that is, $\mathsf{X}^?\varphi$ holds if either there is no following state, or there is a successor state in which φ holds. Let i be an identity term.

$$i : (* = \mathit{false} \Rightarrow \mathsf{X}^?(\mathsf{G}(\mathit{false}))) \tag{1}$$

The following formula expresses that objects and messages can only be destroyed after they existed.

$$i : (* = \mathit{false} \Rightarrow \mathsf{Y}(* = \mathit{true})) \tag{2}$$

In particular, due to the restricted behavior of messages, at most two different states are possible for a message, that is, a message exists and may possibly be destroyed. Therefore, if a message exists, the only possible following state is the one in which it is destroyed. Thus, let m be a message identity term.

$$m : (* = \mathit{true} \Rightarrow \mathsf{X}^?(* = \mathit{false})) \tag{3}$$

Moreover, attributes of objects and messages are functional. Let i be an identity term and let $a = x, a = y$ be attribute terms for i where $x \neq y$.

$$i : (\neg(a = x \wedge a = y)). \tag{4}$$

For objects and messages the attribute $*$ is always defined.

$$i : (\mathit{true} \Rightarrow (* = \mathit{false} \vee * = \mathit{true})). \tag{5}$$

We introduce a special format of DTL^+ formulas, so-called (conditional) rewrite formulas, which will prove useful for the translation of a rewrite theory into a temporal logic theory.

Subsequently, we assume Maude specifications that satisfy all conditions given in Section 3.2. That is, (1) objects and messages are uniquely identified,

(2) creation and deletion of objects and messages is reflected in attributes, and (3) a start rule for initial configurations is defined. As a consequence of (2), all objects and messages of the left-hand side of a rewrite rule appear also in the right-hand side of that rule.

A rewrite rule can be translated into several DTL$^+$ formulas. Each one of these formulas reflects the view of one specific object. A (conditional) rewrite formula is a compact representation for a set DTL$^+$ formulas. Before we give the general definition of a (conditional) rewrite formula, we illustrate the idea with the help of an example. Let $\varphi, \varphi', \psi, \psi'$, and ρ' be formulas which express the state of objects, that is, conjunctions of terms of sort attribute-value pair. The rewrite rule

(6) `< i; C |` φ `> < j; C' |` ψ `>`
`=> < i; C |` φ' `> < j; C' |` ψ' `> < k; C''|` ρ `>`

can be interpreted as a synchronization between the objects `i` and `j`. The result of applying this rule is that `i` and `j` change their state and a new object `k` is created. The resulting state is understood to be distributed, that is, `i`, `j`, and `k` are independent. The information inherent in such a rewrite rule can be formulated from the different viewpoints of the objects `i`, `j`, and `k`. From the viewpoint of object `i` it is true that it may transform from a state φ to a state φ' provided that it synchronizes with object `j` in state ψ. From the viewpoint of object `j` it is true that it may transform from a state ψ to a state ψ' provided that it synchronizes with object `i` in state φ. Note that the newly created object `k` is not part of any of the local formulas. This can be explained as follows. From the local viewpoints of objects `i` and `j` there is no synchronization with object `k` in the current state since `k` does not exist yet, nor in the successor state since the successor states of all objects on the right-hand side are independent.

In this way, a rewrite rule with several objects on the left-hand side represents a synchronization between those objects. In particular, applying this rewrite rule requires the objects of the left-hand side to synchronize to perform a transition to their new local states. But we do not interpret the right-hand side of the rewrite rule as a synchronization. Rather, we emphasize the idea of distributed systems and, therefore, understand the right-hand side of a rule to describe a possible *distributed* successor state. That is, per se the objects on the right-hand side of the rule do not synchronize.

Given this understanding of a rewrite rule, one can derive several formulas in DTL$^+$ from a rewrite rule which express the local views of the objects involved in the rule. From rewrite rule (6) we can derive the following two local formulas:

(7) $i : ((\varphi \wedge j : (\psi)) \Rightarrow \rhd(\varphi'))$,
(8) $j : ((\psi \wedge i : (\varphi)) \Rightarrow \rhd(\psi'))$.

As an abbreviation for formulas (7) and (8) we introduce the following so-called rewrite formula

$$\{i : \varphi,\ j : \psi\} \Rightarrow \triangleright(\{i : \varphi',\ j : \psi'\}).$$

A rewrite formula is a short form for a set of formulas where a fact is expressed from different viewpoints of communicating objects, exploiting all possible permutations on the identities. Since rewrite rules in Maude may be conditional, rewrite formulas may be conditional. A condition in a rewrite formula may only use variables which are part of the left-hand side of the rule.

Definition 4.2 *Let $i_1, \ldots, i_n$ be pairwise distinct identity terms, i.e., each term i_ν is either an object identity term of some class C ($i_\nu \in T_\Sigma(X)_{CId}$) or a message identity term ($i_\nu \in T_\Sigma(X)_{MId}$). Let $i_\nu : \varphi_\nu \in \mathrm{DTL^+}_{i_\nu} (\nu = 1, \ldots, n)$ be well-formed formulas, and let Π be the set of all permutations on $\{1, \ldots, n\}$. Then, a (conditional) rewrite formula*

$$\{i_1 : \varphi_1,\ i_2 : \varphi_2,\ \ldots,\ i_n : \varphi_n, c\} \Rightarrow \triangleright(\{i_1 : \varphi'_1,\ i_2 : \varphi'_2,\ \ldots,\ i_n : \varphi'_n\})$$

for a condition c, where c only uses variables in $\{i_1 : \varphi_1,\ i_2 : \varphi_2,\ \ldots,\ i_n : \varphi_n\}$, is defined to be the set of formulas $\{i_{\pi(1)} : ((\varphi_{\pi(1)} \wedge i_{\pi(2)} : (\varphi_{\pi(2)}) \wedge \ldots \wedge i_{\pi(n)} : (\varphi_{\pi(n)}) \wedge c) \Rightarrow \triangleright(\varphi'_{\pi(1)})) \mid \pi \in \Pi\}$.

4.2 Translating Maude Rules to DTL$^+$: An Example

We illustrate the main ideas by means of the "Communicating Variables" example. For variables $V \in X_{VariableId}$, $S \in X_{SenderId}$, $R \in X_{ReceiverId}$, $O \in X_{OId}$, $N, M \in X_{Nat}$ the following formulas are given by translating the Maude rewrite rules into (conditional) rewrite formulas.

(9) $\{V : (val = 0 \wedge set = N.Set \wedge * = true), N \neq 0\} \Rightarrow$
$\triangleright(\{V : (val = N)\})$,

(10) $\{S : (val = N \wedge rec = O \wedge counter = M \wedge * = true), N \neq 0\} \Rightarrow$
$\triangleright(\{S : (val = 0 \wedge counter = M + 1)\})$,

(11) $\{R : (val = N \wedge * = true)$,
$(S, L) : (cnt = to(R, M) \wedge * = true), N \neq 0\} \Rightarrow$
$\triangleright(\{R : (val = 0),\ (S, M) : (* = false)\})$.

As one can see, newly created objects, that is, objects which appear only on the right-hand side of a rule, have no impact on the rewrite formulas.

More generally, the operator $\mathcal{TL}$ which maps a rewrite theory to a temporal logic theory is given in the following way. Given a rewrite theory $\mathcal{R} = (\Sigma, E, L, R)$ we assume a signature $\Sigma = (S, \Omega)$ with sorts as described in Section 4.1. The temporal logic theory $\mathcal{TL}(\mathcal{R}) = (\Sigma, E, \Phi)$ over a given rewrite theory $\mathcal{R}$ consists of the signature and equations as given in $\mathcal{R}$ and DTL$^+$-formulas Φ given by translating rewrite rules into rewrite formulas plus the predefined frame formulas.

5 Models for DTL^+

We briefly define interpretation structures for DTL^+. For this purpose we adapt interpretation structures for DTL (see [ECSD98,ESSS94]). Then, we provide a construction of a model of a temporal logic theory which is based on the proof terms of the underlying rewrite theory. In this way, we give an event structure semantics to object-oriented Maude modules. The temporal logic theory is the basis for further verification steps.

5.1 Semantics of DTL^+

The reader is referred to [ECSD98,ESSS94] for more detailed information on event structure models for object-oriented systems. Objects and messages are sequential processes which are capable of executing transitions in a sequential manner. An execution of a rewrite rule corresponds to an event. Sequences of events represent a possible execution order of an object or a message. At each state an object or a message may proceed in several ways. Thus, the set of all possible executions has a tree structure. Allowing for several start states, we arrive at a set of trees: a so-called grove. An event grove G is a model for one object or one message and can be understood as a sequential prime event structure $E(G) = (Ev, \rightarrow)$ (cf. [WN95]). Ev is the set of events and $\rightarrow$ is a partial order representing causality between events. A sequential life cycle L is a maximal, totally ordered (conflict-free) trace in G. The maximal event of a sequential, finite life cycle is the one which causally depends on all other events, that is, $e \in L$ maximal iff $\forall e' \in L : e' \rightarrow e$.

For our purposes we assume a unique minimal event $\epsilon \in Ev$ and denote proper events as $Ev^+ = Ev - \{\epsilon\}$. ϵ corresponds to a prenatal state, that is , a state where no rewrite rule occurred so far. To provide interpretation structures, each proper event is labeled with a set of object or message attributes, that is, attribute-value pairs. Given the quotient term algebra $T_{\Sigma,E}$ we are interested in the subalgebra $(I, Att) = (I, \{Att_i\}_{i \in I})$ of congruence classes of identity and attribute terms. A labeling for an event grove G and an object or message i is a total function $\lambda_i : Ev^+ \rightarrow 2^{Att_i}$. Interpretation structures for systems are distributed labeled event groves $(G, \lambda) = (\bigcup_{i \in I} G_i, \{\lambda_i\}_{i \in I})$, where G_i is an event grove. (G, λ) is called a system behavior. Intuitively, a distributed event grove is a family of local event groves that may share events, that is, the local event sets need not be disjoint. A distributed event grove may be considered as a presentation of a prime event structure $E(G) = (Ev, \rightarrow, \#)$. $\#$ is the symmetric, irreflexive conflict relation representing choice. Events which are neither in causal relation nor in conflict relation are called concurrent events. Thus, a sequential prime event structure is a prime event structure without concurrency, that is, $\forall e, f : Ev, \neg(e \rightarrow f \vee f \rightarrow e) \Rightarrow e \# f$ holds in a sequential prime event structure. A distributed life cycle $L = \bigcup_{i \in I} L_i$ in G is the union of sequential life cycles $L_i = \{e \in L \mid e \in Ev_i\}$, i.e., maximal traces in G_i. For a given rewrite theory $\mathcal{R} = (\Sigma, E, L, R)$, the logic is interpreted over the corresponding subalgebra (I, Att) of $T_{\Sigma,E}$, a tuple (B, L), where $B = (G, \lambda)$ is a system behavior, and a distributed life cycle in G: $L \in \mathcal{L}(G)$. In particular, formulas are interpreted for a variable assignment θ in a local event $e \in Ev^+$. Data terms are to be interpreted globally in $T_{\Sigma,E}$. The satisfaction relation is very similar to the one presented in [ECSD98].

Given this semantics one can show that a rewrite formula as introduced in Section 4.1 is an abbreviation for a set of formulas which are all equivalent.

Proposition 5.1 *All formulas in* $\Psi := \{i_1 : \varphi_1,\ i_2 : \varphi_2,\ \ldots,\ i_n : \varphi_n, c\} \Rightarrow \triangleright(\{i_1 : \varphi'_1,\ i_2 : \varphi'_2,\ \ldots,\ i_n : \varphi'_n\})$ *are equivalent, i.e., for a given system behavior* $B = (G, \lambda)$ *and arbitrary, but fixed formulas* $\psi, \psi' \in \Psi$ *it holds that* B *satisfies* ψ *iff* B *satisfies* ψ'.

5.2 A Model Construction

Given a rewrite theory $\mathcal{R} = (\Sigma, E, L, R)$, the initial model is a category $\mathcal{T}_{\mathcal{R}}(X)$ whose objects are equivalence classes of terms $[t] \in T_{\Sigma,E}(X)$ and whose morphisms are equivalence classes of proof terms representing proofs in rewriting deduction, that is, concurrent $\mathcal{R}$-rewrites (for details see, for instance, [Mes92,Mes93]). We intend to construct a model from those proof terms which start with the application of the `start` rule. We assume that all possible initial configurations are ground terms and that all rewrite rules do not introduce new variables on the right-hand side. Therefore, all proof terms will transform between congruence classes of ground terms in $T_{\Sigma,E}$. Moreover, we make some assumptions about the nature of configurations. We assume so-called coherent configurations. In a coherent configuration, identities of objects and messages are unique. We require that an application of a rewrite rule to a coherent configuration results in a coherent configuration. Moreover, we assume that a rewrite step in a proof term is unique (as, for instance, done by introducing the identifying parameter in our example). We will exploit this by using the rewrite steps to give unique names to the events of the model. The ideas for coherent configurations are borrowed from [MT98]. Let $\mathcal{T}^c_{\mathcal{R}}$ be the substructure of the initial model $\mathcal{T}_{\mathcal{R}}(X)$ in which proof terms satisfy the above conditions.

The first partial order semantics for concurrent objects has been presented in [MT98]. Meseguer and Talcott assume object-oriented specifications which satisfy the conditions for coherent configurations. The application of a proof term to a given start configuration involves several rewrites and ultimately leads to another configuration. In [MT98] a partial order of rewrite events is constructed over the structure of a proof term. The result is a category whose objects are coherent configurations and whose morphism are partial orders of events. Meseguer and Talcott show that two proof terms in $\mathcal{T}^c_{\mathcal{R}}(X)$ are provably equivalent if and only if their associated partial orders are the same. There are some differences to our work. First, we explicitly introduce messages as a special kind of objects to treat objects and messages homogeneously in our framework. Second, we construct sequential models for objects and messages which reflect the local viewpoints of them, that is, we do not assume a global configuration as done in [MT98]. In the future we will investigate the exact relation between the event structure model presented here and the model given in [MT98].

Subsequently, we refer to (I, Att) as being the subalgebra of $T_{\Sigma,E}$ of congruence classes of identity and attribute terms. Because of the restrictions we make on proof terms, all coherent configurations which appear in proof terms, will be ground. For such configurations we define two functions: a function ids which maps a coherent, ground configuration to a set of identity terms and for each identity term i a function l_i which maps to a set of attribute-value pairs. $ids : CConf \to 2^{(\bigcup_C T_{\Sigma,CId} \cup T_{\Sigma,MId})}$ gives identity terms for a coherent, ground configuration. For a given coherent, ground configuration C, we refer to the elementwise interpretation of identity terms as $ids^{T_{\Sigma,E}}(C) = \{[i] \mid i \in ids(C)\} \in I$. ids is extended to a function on rewrites $r(\theta)$ by applying ids to the coherent, ground configurations on both sides of the rule. Moreover, we define a labeling function l_i which for a given coherent, ground configuration C gives the set of all attribute-value terms of the object with identity i. Thus, $l_i : CConf \to 2^{T_{\Sigma,CAtt}}$ if $i \in T_{\Sigma,CId}$ or $\lambda_i : CConf \to 2^{T_{\Sigma,MAtt}}$ if $i \in T_{\Sigma,MId}$, respectively. We also interpret attribute-value pairs in the quotient term algebra. For a given coherent, ground configuration C and an identity term i, $l_i^{T_{\Sigma,E}}(C) \in Att_{[i]}$ is the set of congruence classes of attribute-value terms.

The idea is to build the system model as a set of distributed life cycles. Each proof term determines a distributed life cycle. Proof terms are of the following form:

$$\frac{C \xrightarrow{\alpha} C' \xrightarrow{\beta} C''}{C \xrightarrow{\alpha;\beta} C''}, \quad C, C', C'' : CConf$$

$$\frac{C \xrightarrow{\alpha} C' \quad D \xrightarrow{\beta} D'}{CD \xrightarrow{\alpha\beta} C'D'}, \quad C, D, CD : CConf$$

$$\frac{C}{C \xrightarrow{C} C}, \quad C : CConf$$

$$\frac{r : C \longrightarrow C' \text{ in } R}{r(\theta) : \theta(C) \longrightarrow \theta(C')}, \quad \theta \text{ substitution}, C, \theta(C) : CConf$$

Before we go into details concerning the model construction we will illustrate the ideas with the help of our example. Given the extended Maude specification in Section 3.2, one possible proof term is

```
start([C <- initial]);
(choose_and_change([V <- s, N <- 1, Set <- 2.3])
 choose_and_change([V <- r, N <- 1, Set <- 2.3]));
```

`send([S <- s, N <- 1, R <- r, M <- 1])`);

`receive([R <- r, N <- 1, S <- s, L <- 1, R <- r, M <- 1])`).

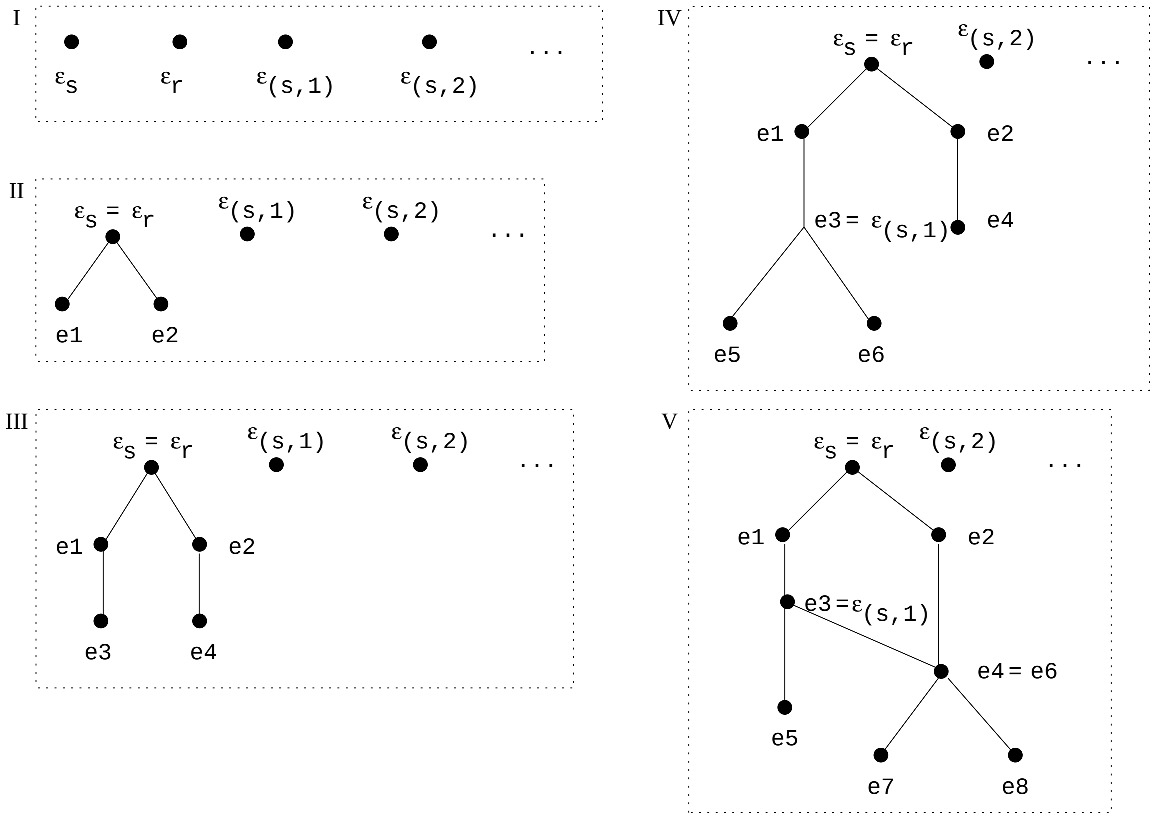

Fig. 2. Construction of part of a distributed life cycle.

In Figure 2 we illustrate the four steps in building a distributed life cycle from the above proof term. The first box illustrates the prenatal life cycle. The distributed life cycle in the second box is the result of applying the rule `start([C <- initial])`. In the third box the distributed life cycle which is the result of the concurrent application of both `choose_and_change` rules is depicted.

We have abbreviated the event names in the figure. More precisely, the names are derived from the rule names, the substitution, and the object, that is,

`e1=start([C <- initial])`$_{\mathtt{s}}$,

`e2=start([C <- initial])`$_{\mathtt{r}}$,

`e3=choose_and_change([V <- s, N <- 1, Set <- 2.3])`$_{\mathtt{s}}$,

`e4=choose_and_change([V <- r, N <- 1, Set <- 2.3])`$_{\mathtt{r}}$,

`e5=send([S <- s, N <- 1, R <- r, M <- 1])`$_{\mathtt{s}}$,

`e6=send([S <- s, N <- 1, R <- r, M <- 1])`$_{\mathtt{(s,1)}}$,

`e7=receive([R <- r, N <- 1, S <- s, L <- 1, R <- r, M <- 1])`$_{\mathtt{(s,1)}}$,

`e7=receive([R <- r, N <- 1, S <- s, L <- 1, R <- r, M <- 1])`$_{\mathtt{r}}$.

In the distributed life cycle in the fifth box the following are examples of pairs of concurrent events: `(e1, e2)`, `(e3, e2)`, `(e4, e5)`, `(e5, e7)`, `(e7, e8)`. Moreover, the following holds

$$\texttt{e1, e3, e5} \in Ev_{\texttt{s}},$$
$$\texttt{e2, e4, e7} \in Ev_{\texttt{r}},$$
$$\texttt{e6, e7} \in Ev_{(\texttt{s},\texttt{1})}.$$

The general algorithm for constructing a distributed life cycle from a proof term is as follows.

Definition 5.2 *Let $\mathcal{R} = (\Sigma, E, L, R)$ be a rewrite theory and let $T_{\Sigma,E}$ be the quotient term algebra. Let (I, Att) be the subalgebra of $T_{\Sigma,E}$ of congruence classes of identity and attribute terms. Let $\mathcal{T}^c_{\mathcal{R}}$ be the substructure of the initial model $\mathcal{T}_{\mathcal{R}}(X)$ in which all proof terms start with the application of the start rule and transform between ground, coherent configurations. Let $ids^{T_{\Sigma,E}}$ and $l_i^{T_{\Sigma,E}}$ be functions as defined above. Let $L_{prenatal} = (\{\epsilon_i \mid i \in I\}, \emptyset, \emptyset)$ be the prenatal life cycle. From a proof term α we construct a distributed (finite) labeled life cycle $L_\alpha = (Lv, \rightarrow, \lambda)$ as the result of $BuildDLC(L_{prenatal}, \alpha)$. For a labeled life cycle $L = (Lv, \rightarrow, \lambda)$, $BuildDLC(L, \alpha)$ is inductively defined over the structure of the proof term α:*

- *$\alpha = C \xrightarrow{C} C$: return L.*
- *$\alpha = r(\theta) : \theta(C) \longrightarrow \theta(C')$: Let $ids(r(\theta))^{T_{\Sigma,E}} = \{[i_1], \ldots, [i_n]\}$ be the set of congruence classes of identity terms of objects which are involved in the rule. With $e^{max}_{[i_\nu]}, \nu = 1, \ldots, n$ we denote the maximal events in the current life cycle of those objects which are involved in the rule. We identify all these maximal events to express the need for synchronization, i.e., $e^{max} := e^{max}_{[i_\nu]} = e^{max}_{[i_\mu]}, \nu, \mu = 1, \ldots, n$. Names for the new events are constructed by using the identity, the rule, and the substitution, i.e., $r_{[i_\nu]}(\theta)$.*
  Return $(Lv \cup \bigcup_\nu r_{[i_\nu]}(\theta), \rightarrow \cup \bigcup_\nu (e^{max} \rightarrow_{[i_\nu]} r_{[i_\nu]}(\theta)), \lambda')$, where λ' extends λ by $\lambda_{[i_\nu]}(r_{[i_\nu]}(\theta)) = l_{i_\nu}^{T_{\Sigma,E}}(\theta(C'))$ $(\nu = 1, \ldots, n)$.
- *$\alpha = \beta; \gamma$: return $BuildDLC(BuildDLC(L, \beta), \gamma)$.*
- *$\alpha = \beta\gamma$: return $BuildDLC(L, \beta) \cup BuildDLC(L, \gamma)$.*

Let $L_\alpha = BuildDLC(L_{prenatal}, \alpha)$ be the distributed labeled life cycle constructed from a proof term α. Then, the finite system behavior $\mathcal{B}^{fin} = (G^{fin}, \lambda^{fin})$ is the *disjoint* union of all distributed life cycles constructed from proof terms in $\mathcal{T}^c_{\mathcal{R}}$. The system behavior $\mathcal{B}(\mathcal{T}^c_{\mathcal{R}})$ is given by an infinite extension of finite behaviors, such that each finite sub-behaviors of an infinite behavior is constructed from a proof term in $\mathcal{T}^c_{\mathcal{R}}$.

Definition 5.3 *Let $\mathcal{R} = (\Sigma, E, L, R)$ be a rewrite theory and let $\mathcal{T}^c_{\mathcal{R}}$ be the substructure of the initial model $\mathcal{T}_{\mathcal{R}}(X)$ in which all proof terms start with the application of the start rule and transform between ground, coherent configurations. For each proof term $\alpha \in \mathcal{T}^c_{\mathcal{R}}$ we construct a distributed labeled life cycle $L_\alpha = (Lv_\alpha, \rightarrow_\alpha, \lambda_\alpha)$ as given in Def. 5.2. Then, the union of all distributed life*

cycles $\mathcal{B}^{fin}(\mathcal{T}^c_{\mathcal{R}}) = \bigcup_\alpha L_\alpha$ *is a finite system behavior,* $\mathcal{B}^{fin}(\mathcal{T}^c_{\mathcal{R}}) = (G^{fin}, \lambda^{fin})$, $G^{fin} = (Ev, \rightarrow)$, $Ev = \bigcup_\alpha Lv_\alpha$, $\lambda^{fin} = \bigcup_\alpha \lambda_\alpha$. *For events* $e, e' \in Ev_i$ *for some* $i \in I$ *it holds that* $e \# e'$ *if and only if they are not causally ordered.*
The system behavior $\mathcal{B}(\mathcal{T}^c_{\mathcal{R}}) = (G, \lambda)$ *is the infinite extension of finite behaviors, where for each finite part of a distributed life cycle* $L \in \mathcal{L}(G)$ *it must be true that* L *can be constructed from a proof term* $\alpha \in \mathcal{T}^c_{\mathcal{R}}$.

This construction is sound.

Theorem 5.4 *Given a temporal logic theory* $\mathcal{TL}(\mathcal{R}) = (\Sigma, E, \Phi)$ *which is the result of transforming an extended Maude specification, and given a subalgebra* (I, Att) *of* $T_{\Sigma,E}$, $\mathcal{B}(\mathcal{T}^c_{\mathcal{R}})$ *is a system behavior over* (I, Att) *which satisfies* Φ.

6 Concluding Remarks

A temporal logic framework can be applied to give semantics to Maude object-oriented specifications. This framework constitutes the basis for formal reasoning about Maude, exploiting the expressiveness and techniques of temporal logic. In the future, we will investigate case studies and work on an axiomatization of DTL^+. Because of the reflective properties of rewriting logic, one can easily specify different formalisms in rewriting logic. Different formal method tools have already been designed and implemented. We intend to exploit this fact for the temporal logic framework. Moreover, we will investigate whether the mapping from rewrite theories to temporal logic theories is a map of entailment systems or even a map of logics in the sense of [MOM94].

Acknowledgments. I am especially indebted to José Meseguer who discussed with me the ideas presented in this paper. I have benefited very much from conversations with him, Narciso Martí-Oliet, Carolyn Talcott, Tom Maibaum, Jose Fiadeiro, and Peter Ölveczky. The efforts of Hans-Dieter Ehrich to develop a distributed temporal logic for synchronous systems were a starting point for this work. Many thanks to Francisco Durán who remained unflustered by my frequent questions about Maude and rewriting logic. I am grateful to Francisco Durán and Narciso Martí-Oliet who gave valuable comments on an earlier draft of this paper.

References

[AH92] R. Alur and T. Henzinger. Logics and Models of Real Time: A Survey. In *Real Time: Theory in Practice, Mook, The Netherlands, June 1991*, pages 74–106. Springer, 1992. LNCS 600.

[CDELM98] M. Clavel, F. Durán, S. Eker, P. Lincoln, and J. Meseguer *An Introduction to Maude (Beta Version)*. Manuscript, SRI International, Computer Science Laboratory, Menlo Park, CA, 1998.

[DM98] F. Durán and J. Meseguer. An Extensible Module Algebra For Maude. This volume.

[ECSD98] H.-D. Ehrich, C. Caleiro, A. Sernadas, and G. Denker. Logics for Specifying Concurrent Information Systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 167–198. Kluwer Academic Publishers, 1998.

[ESSS94] H.-D. Ehrich, A. Sernadas, G. Saake, and C. Sernadas. Distributed Temporal Logic for Concurrent Object Families. In R. Wieringa and R. Feenstra, editors, *Working papers of the International Workshop on Information Systems - Correctness and Reusability*, pages 22–30. Vrije Universiteit Amsterdam, Rapport Nr. IR-357, 1994.

[KW95] P. Kosiuczenko and M. Wirsing. Timed Rewriting Logic for the Specification of Time-Sensitive Systems. In H. Schwichtenberg, editor, *Proc. of the Intern. Summer School on Proof and Computation.* Springer, 1995. NATO-ASI Series.

[Lec96] U. Lechner. Object-oriented specifications of distributed systems in the μ-calculus and maude. In Meseguer [Mes96a], pages 384–403.

[Lec97] U. Lechner. *Object-Oriented Specification of Distributed Systems.* PhD thesis, University of Passau, 1997. Appeared as Technical Report, Fakultät für Mathematik und Informatik, Universität Passau, MIP-9717. Available at: `www.mcm.unisg.ch/~ulechner`.

[Mes92] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[Mes93] J. Meseguer. A Logical Theory of Concurrent Objects and Its Realization in the Maude Language. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 314–390. The MIT Press, 1993.

[Mes96a] J. Meseguer, editor. *Rewriting Logic and Its Applications, First International Workshop, Asilomar Conference Center, Pacific Grove, CA, September 3-6, 1996.* Elsevier Science B.V., Electronic Notes in Theoretical Computer Science, Volume 4, `http://www.elsevier.nl/locate/entcs/volume4.html`, 1996.

[Mes96b] J. Meseguer. Rewriting Logic as a Semantic Framework for Concurrency: A Progress Report. In U. Montanari and V. Sassone, editors, *Proc. 7th Intern. Conf. on Concurrency Theory: CONCUR'96, Pisa, August 1996*, pages 331–372, 1996. LNCS 1119.

[MOM94] N. Martí-Oliet and J. Meseguer. General Logics and Logical Frameworks. In D. Gabbay, editor, *What is a Logical System?*, pages 355–392. Oxford University Press, 1994.

[MT98] J. Meseguer and C. Talcott. Partial Order Semantics for Concurrent Objects. Manuscript, SRI International and Stanford University Computer Science Department, April 1998.

[ÖM96] P. Ölveczky and J. Meseguer. Specifying Real-Time Systems in Rewriting Logic. In Meseguer [Mes96a], pages 283–308.

[WN95] G. Winskel and M. Nielsen. Models for Concurrency. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Vol. 4, Semantic Modelling*, pages 1–148. Oxford Science Publications, 1995.