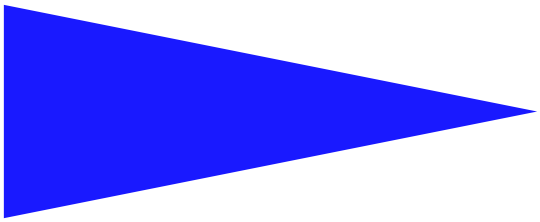
Theorem Proving for Maude's Rewriting Logic

Vlad Rusu, Manuel Clavel

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1873



THEOREM PROVING FOR MAUDE'S REWRITING LOGIC

VLAD RUSU AND MANUEL CLAVEL

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Theorem Proving for Maude's Rewriting Logic

Vlad Rusu and Manuel Clavel*

Systèmes communicants
Projet VerTeCs

Publication interne n ° 1873 — Novembre 2007 — 45 pages

Abstract: We present an approach based on inductive theorem proving for verifying invariance properties of systems specified in Rewriting Logic, an executable specification language implemented (among others) in the Maude tool. Since theorem proving is not directly available for rewriting logic, we define an encoding of rewriting logic into its membership equational (sub)logic. Then, inductive theorem provers for membership equational logic, such as the ITP tool, can be used for verifying the resulting membership equational logic specification, and, implicitly, for verifying invariance properties of the original rewriting logic specification. The approach is illustrated first on a 2-process Bakery algorithm and then on a parameterised, n-process version of the algorithm.

Key-words: Rewriting logic, inductive theorem proving, Maude

(Résumé : tsvp)

* Universidad Complutense de Madrid, Facultad de Informática, Departamento de Sistemas Informáticos y Programación, MADRID, Spain

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Preuve de théorèmes pour la logique de réecriture

Résumé : Nous présentons une approche basée sur la preuve inductive de théorèmes pour vérifier des propriétés d'invariance de systèmes décrits en logique de réécriture, un langage de spécifications exécutables implémemté dans des outils tel que Maude. La preuve inductive de théorèmes n'existant pas pour la logique de réécriture, nous définissons un codage de cette logique dans la logique équationnelle "avec appartenance". Ceci nous permet d'utiliser des prouveurs inductifs tels que l'outil ITP sur la théorie équationnelle ainsi produite, et, implicitement, de prouver des propriétés d'invariance sur la spécification en logique de rééecriture de départ. L'approche est illustrée dans un premier temps sur l'algorithme du boulanger à deux processus, et ensuite sur une version paramétrique de cet algorithme impliquant n processus.

Mots clés : logique de réécriture, preuve inductive de théorèmes, Maude

1 Introduction

Rewriting logic [1] is a formal executable specification language. Several tools implement several variants of the logic, e.g., Maude [2], Elan [3], and CAFEOBJ [4]. In rewriting logic (often abbreviated as RL in this paper) a system's dynamics is expressed by means of rewrite rules, and the system's data is expressed equationally, in some version of equational logic. For example, Maude's version of RL, which we consider here, contains as a sublogic the *membership equational logic* (MEL) [5], which generalises order-sorted logic with so-called *membership assertions*. A membership assertion simply states the membership of a term in a sort; such assertions are quite expressive, allowing one to define nontrivial sorts such as *the sort of terms reachable from an initial term* in a rewriting logic specification [6].

The Maude system [2] consists of a language for conveniently expressing RL and MEL specifications, and a set of tools for analysing such specifications and verifying them against user-defined properties. The *finite-state* verification tools include a state-space searching tool and a model checker for Linear Temporal Logic properties [7]. *Infinite-state* systems can also be verified, by abstracting infinite state spaces to finite-state ones using *equational abstractions* [8] in conjunction with model checking. However, coming up with adequate abstractions expressible as a set of equations is not always easy. *Interactive theorem proving* remains a useful verification approach for systems with infinite or large state spaces.

In this paper we propose interactive theorem-proving techniques for verifying *invariance properties* of rewriting logic specifications, i.e., properties that hold on all terms reachable from a given initial term. The work is based on the existing tool: ITP (Inductive Theorem Prover, developed by the second author [9]) dedicated to the MEL sublogic of RL. Rather than extending ITP from MEL to RL, we chose to take an opposite approach: we encode RL into MEL, and we use the existing ITP tool for verification.

We note that our approach allows us to go beyond Maude's implementation of Rewriting Logic; that is, we can verify some RL specifications that are not *executable* in Maude, due to supplementary variables in the right-hand sides and conditions of rules. This allows us to deal with reactive systems composed of a parametric or even a variable number of processes (allowing for process creation/destruction). We illustrate the approach on an example that has some of these features: an n-process mutual exclusion Bakery algorithm.

The rest of this paper is organised as follows. In Sections 2 and 3 we present some background notions on membership equational logic and rewriting logic. We define the notion of an *invariant φ of a* RL *theory $\mathcal{R}$ starting from an initial term t_0*, denoted by $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$. Note that the term t_0 is not necessarily ground, which allows for describing possibly infinitely many initial states. For a predicate φ on ground terms, $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$ means *$\varphi(t)$ is provable in the initial model of the* MEL *sub-theory of $\mathcal{R}$, for all ground terms t reachable from t_0 by top-level rewriting*[1] *in $\mathcal{R}$*. We justify why this is an adequate definition of invariant in our context. However, the definition uses concepts from RL, hence, it is not usable by our theorem prover ITP, which only "speaks" MEL.

[1] Top-level rewriting is a restricted form of rewriting that prohibits rewriting on strict subterms. It is shown in [10] to be enough for specifying a wide class of dynamic systems.

The core of the approach is presented in Section 4. First, an automatic translation takes a RL theory $\mathcal{R}$ and a term t_0, and generates a MEL theory $\mathcal{M}(\mathcal{R}, t_0)$, which enriches the MEL subtheory of $\mathcal{R}$ with a new sort called *Reachable* and with membership assertions inductively defining this sort. Then, we prove that "being of sort *Reachable* in $\mathcal{M}(\mathcal{R}, t_0)$" and "being reachable in $\mathcal{R}$ from t_0 by top-level rewriting" are equivalent statements. Next, we give an alternative definition for invariance, denoted by $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi$, and meaning that *in the initial model of* $\mathcal{M}(\mathcal{R}, t_0)$, $\varphi(t)$ *holds for all terms* t *of sort Reachable*. We prove that the two definitions of invariance: $\vdash$ and $\vdash_{ind}$ are equivalent; the advantage of the second definition is that it uses only MEL concepts. Finally, proving $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$ amounts to proving $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi$.

We then give guidelines for performing such proofs. Typically, one first attempts to prove the statement $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi$ automatically, using induction, rewriting, and decision procedures. When this is not enough, the user examines the subgoals left unproved as well as the ITP *context*, and is usually able to create auxiliary lemmas, which are also invariance statements and must first be proved as (recursively) described here. Then, the lemmas are used to close the pending subgoals in proof of the main invariance statements and, eventually, to complete the proof. The approach is illustrated in Section 5, and Section 6 concludes. For better readability, most proofs of the results are presented in a separate Appendix.

Related Work

In [6], Bruni and Meseguer present a translation of all Rewriting Logic into Membership Equational Logic. Their translation allows for conditions that include rewrites in the conditions of rewrite rules and general rewriting (non only top-level); and they encode reachability between arbitrary terms. By contrast, we do not handle rewrites in conditions and are restricted to top-level rewriting, and we encode reachability from a given initial term. But we also encode *invariance properties*, which [6] does not consider. Another difference is in the complexity of the encoding: [6] uses four different sorts for all the kinds in the MEL theory underlying the rewrite theory to be translated, and several operations on them, whereas we need only one supplementary sort and kind, and no supplementary operation. Finally, perhaps the strongest argument in favor of our translation is that it induces a working approach that allows for practical verification of practically interesting (invariance) properties.

Related logical approaches that allow for proving behavioural properties includes Goguen and Rosu's hidden logic [11], incorporated in the CafeOBJ toolset version of rewriting logic [4]. These logics include so-called *hidden sorts*; a typical example of a hidden sort is the state of the system, which is not observable as such, but can only be (partially) observed via certain operations called *observers*. The internal evolution of a state can be also described using some specific operations. CafeOBJ includes a theorem prover, which has recently been used for verifying an electronic purse system [12]. The methodology advocated in that paper advocates the use of Maude for simulation and debugging, and of the CafeOBJ prover for proving properties. Our approach is more tightly integrated, as we remain inside the Maude environment.

2 Membership Equational Logic

We start in Section 2.1 with a brief introduction to membership equational logic; for a full account the reader may consult [5]. In Section 2.2 we give some results related to the initial model of a MEL theory, and in Section 2.3 we study the deduction of membership assertions.

2.1 Basic Definitions

A membership equational logic (MEL) *signature* is a triple (K, Σ, S) where K is a set of *kinds*, Σ is a $K^* \times K$ indexed family of function symbols $\Sigma = \{\Sigma_{w,k}\}_{(w,k)\in K^*\times K}$, and $S = \{S_k\}_{k\in K}$ is a pairwise disjoint K-indexed family of sets of *sorts* - where S_k is the set of sorts of kind k. A signature (K, Σ, S) is often denoted simply by Σ; then, T_Σ denotes the set of ground terms over signature Σ. Given a set $X = \{x_1 : k_1, \ldots, x_n : k_n\}$ of *kinded variables*, $T_\Sigma(X)$ denotes the set of terms with free variables in the set X. Similarly, $T_{\Sigma,k}$ and $T_{\Sigma,k}(X)$ respectively denote the set of ground terms of kind k (resp. the set of terms of kind k with free variables in the set X). A MEL *atomic formula* over Σ is either an equality $(\forall X)t = t'$ between two terms in $T_{\Sigma,k}(X)$ of the same kind k - the notation $(\forall X)$ emphasises the fact that free variables are universally quantified - or a *membership assertion* $(\forall X)t : s$, where the term $t \in T_{\Sigma,k}(X)$ and $s \in S_k$ is a sort in the set of sorts of kind k. A MEL *sentence* is a universally quantified Horn sentence on atomic formulas, i.e., for $t, t' \in T_\Sigma(X)$, a sentence of the form

$$(\forall X)t = t' \text{ if } C, \text{ or} \tag{1}$$

$$(\forall X)t : s \text{ if } C \tag{2}$$

where the *condition* C has the form (for some finite sets of indices I, J):

$$\bigwedge_{i\in I}(\forall X)(u_i = v_i) \wedge \bigwedge_{j\in J}(\forall X)(w_j : s_j)$$

Sentences of the form (1) are called *conditional equations*, and sentences of the form (2) are called *conditional memberships*. The equations, resp. memberships are *unconditional* when the sub-sentence (... *if* C) is absent. We shall often associate *labels* to membership assertions, e.g., $(\mu) : (\forall X)t : s \text{ if } C$, where μ is a label uniquely identifying the assertion, and often refer to membership assertions *via* their labels.

A MEL *theory* is a tuple $\mathcal{M} = (\Sigma, E)$ that consists of a MEL signature Σ and a set of MEL sentences over Σ. As an example, consider the fragment of specification for natural numbers given in Figure 1, written in the (mostly self-explanatory) Maude syntax. Worth noting are the following facts:

- sorts are declared explicitly, but *kinds* are not; in the specification given in Figure 1, there is only one kind, denoted by `[Nat]` (or equivalently `[NzNat]`), which, intuitively, contains all terms of sorts `[Nat]` and possibly more terms that have a kind but not a sort (such terms should be thought of as *error terms*, e.g., `0 - (S 0)` if we had defined a substraction operator on natural numbers).

```
fmod NAT is
sorts Nat NzNat .           --- declares two sorts Nat  and NzNat
subsort NzNat < Nat .       --- declares a subsorting  relation
op 0 : -> Nat [ctor] .      --- declares two constructors  0 and S
op s :  Nat -> NzNat [ctor] .
op _+_ : Nat Nat -> Nat .   --- declares an infix operator +
eq 0 +  (n:Nat) = (n:Nat) . --- defines the + operator
eq (n:Nat) + S (m:Nat) = S ((n:Nat) + (m:Nat)) .
endfm
```

Figure 1: Natural numbers in Membership Equational Logic.

- the subsorting relation `NzNat < Nat` is syntactical sugar for the conditional membership `cmb (n:[Nat]): Nat if (n:[Nat]): NzNat`. It means basically that all terms of sort `NzNat` also have sort `Nat`.
- operators are defined on *sorts*, not on kinds as required by the definition of MEL; a sort-level declaration `_+_ : Nat Nat -> Nat` is syntactical sugar for the kind-level declaration `_+_ : [Nat] [Nat] -> [Nat]` together with a conditional membership `cmb (n:[Nat]) + (m:[Nat]) : Nat if (n:[Nat]) : Nat` $\wedge$ `(m:[Nat]) : Nat`.

Semantics. A Σ-algebra A for a MEL theory (K, Σ, S, E) consists of a set A_k for each $k \in K$, of a function $A_f : A_{k_1} \times \ldots A_{k_n} \mapsto A_k$ for each operator $f : k_1 \times \ldots k_n \mapsto k$, and a subset $A_s \subseteq A_k$ for each $s \in S_k$ and $k \in K$. A *valuation* η maps each kinded variable $x : k \in X$ to an element $\eta(x) \in A_k$, and has a natural homomorphic extension to terms (also denoted by η). An algebra A and a valuation η *satisfy* an equation $(\forall X)t = t'$, denoted by $A, \eta \models (\forall X)t = t'$, if $\eta(t) = \eta(t')$. Similarly, A and η *satisfy* a membership $(\forall X)t : s$, denoted by $A, \eta \models (\forall X)t : s$ if $\eta(t) \in A_s$. For conditional equations, satisfaction is defined by $A, \eta \models (\forall X)t = t'$ *if* C, when $\eta(t) = \eta(t')$ *holds whenever* $\eta(C)$ *evaluates to* true[2]. Similarly, for conditional memberships, $A, \eta \models (\forall X)t : s$ *if* C holds when $\eta(t) \in A_s$ *evaluates to true whenever* $\eta(C)$ *evaluates to* true. Then, for an algebra A and a sentence φ, we say that φ is valid on A, denoted by $A \models \varphi$, if $A, \eta \models \varphi$ for all valuations η. When $A \models \varphi$ we also say that A is a model of φ. Finally, A is a model of a MEL theory (Σ, E) if A is a model of all the sentences in E.

Definition 1 *Given a* MEL *theory* $\mathcal{M} = (\Sigma, E)$ *and an atomic formula* e $(\forall X)t = t'$ *or* $(\forall X)t : s$*, we say that* $\mathcal{M}$ *entails* e*, denoted by* $\mathcal{M} \vdash e$ *(or* $E \vdash e$ *when the signature is clear from the context) if* e *can be obtained from the sentences in* E *by using the following rules:*

1. *Reflexivity:* $\frac{t \in T_\Sigma(X)}{E \vdash (\forall X)t = t}$
2. *Membership:* $\frac{E \vdash (\forall X)t' = t \quad E \vdash (\forall X)t : s}{E \vdash (\forall X)t' : s}$

[2] for a condition $C : \bigwedge_{i \in I}(\forall X)(u_i = v_i) \wedge \bigwedge_{j \in J}(\forall X)(w_j : s_j)$, $\eta(C)$ is a shortcut notation for $\bigwedge_{i \in I}(\eta(u_i) = \eta(v_i)) \wedge \bigwedge_{j \in J}(\eta(w_j) \in A_{s_j})$.

3. *Symmetry:* $\frac{E\vdash(\forall X)t'=t}{E\vdash(\forall X)t=t'}$

4. *Transitivity:* $\frac{E\vdash(\forall X)t_1=t_2 \quad E\vdash(\forall X)t_2=t_3}{E\vdash(\forall X)t_1=t_3}$

5. *Congruence:*
$$\frac{f\in\Sigma_{k_1,\ldots,k_n,k} \quad t_1,\ldots t_n\in T_{\Sigma_{k_i}}(X) \quad \text{for } i\in[1,n] \quad t'_i\in T_{\Sigma_{k_i}}(X) \quad E\vdash(\forall X)t_i=t'_i}{E\vdash(\forall X)f(t_1,\ldots,t_i,\ldots,t_n)=f(t_1,\ldots,t'_i,\ldots,t_n)}$$

6. *Replacement*$_1$ $\frac{(\forall X)t=t' \text{ if } C \quad \in E \quad \sigma:X\mapsto T_\Sigma(Y) \quad E\vdash(\forall Y)C\sigma}{E\vdash(\forall Y)t\sigma=t'\sigma}$

7. *Replacement*$_2$ $\frac{(\mu)\ (\forall X)t:s \text{ if } C \quad \in E \quad \sigma:X\mapsto T_\Sigma(Y) \quad E\vdash(\forall Y)C\sigma}{E\vdash(\forall Y)t\sigma:s}$

where $\sigma : X \mapsto T_\Sigma(Y)$ *is a kind-preserving substitution, and, for* $C : \bigwedge_{i\in I}(\forall X)(u_i = v_i) \wedge \bigwedge_{j\in J}(\forall X)(w_j : s_j)$ *and* $\sigma : X \mapsto T_\Sigma(Y)$, $E \vdash (\forall Y)C\sigma$ *is a shortcut for* $\bigwedge_{i\in I} E \vdash (\forall Y)(u_i\sigma = v_i\sigma) \wedge \bigwedge_{j\in J} E \vdash (\forall Y)(w_j\sigma : s_j)$. □

The above deduction system of MEL is complete [5], in the sense that an atomic formula is provable from the sentences of a theory (Σ, E) if and only if it is valid on *all* the models of that theory. However, validity in all models of a given theory is not a very adequate notion of *truth*; for example, the addition of natural numbers is not associative in all models, but only in some models including the so-called *initial model* [5] - more about this in Section 2.2.

Reasoning by induction is a key tool for proving properties on initial models. For this, theorem provers like ITP [9] use induction principles automatically derived a property's definition in a MEL theory. Let us go back to our definition of natural numbers given in Figure 1. For proving by induction that a predicate P holds true for all natural numbers, ITP generates, from the signature of our theory and from the predicate P, an induction principle which basically says that if the predicate holds on all constructors of sort `Nat` then it holds for all terms in the sort. Here, the predicate generated for P is the usual induction principle for natural numbers. Then, e.g., associativity of addition can be stated, informally, as $\forall n.\, P(n) = true$, with $P(n) = \forall m, p.\, (n + m) + p = n + (m + p)$; the induction principle generated by ITP entails the obligations $P(0) = true$ and $\forall n.P(n) = true \Rightarrow P(n+1) = true$.

The ITP tool has been enhanced with decision procedures that automatically discharge proof obligations written in Presburger arithmetic [13]. For instance, the base and inductive steps in the proof of associativity of addition, or even the associativity itself, can be proved automatically.

2.2 Properties of Initial Models

We define the initial model of a MEL theory, and prove some results about initial models that will be used in proving the correctness of our approach.

Definition 2 *The* initial model *of a* MEL *theory* (K, Σ, S, E) *is defined as follows [5]:*

- *for each kind $k \in K$, the set A_k is the set of equivalence classes of ground terms of kind k* modulo *the sentences in E, i.e., $A_k = T_{\Sigma,k/E} = \{[t]_E | t \in T_{\Sigma,k}\}$, where $[t]_E = [t']_E$ iff $E \vdash (\forall \emptyset) t = t'$;*
- *for each function symbol $f : k_1 \times \ldots \times k_n \mapsto k \in \Sigma$, the function $A_f : A_{k_1} \times, \ldots, A_{k_n} \mapsto A_k$ is defined by: $\forall [t_1]_E \in T_{\Sigma,k_1/E}, \ldots, \forall [t_n]_E \in T_{\Sigma,k_1/E}$: $A_f([t_1]_E, \ldots, [t_n]_E) = [f(t_1, \ldots, t_n)]_E$, where the definition does not depend on the particular terms $t_1, \ldots, t_n$ by virtue of the* Congruence *rule in Definition 1;*
- *for each $k \in K$ and sort $s \in S_k$, the set A_s is the set of equivalence classes of terms $A_s = T_{\Sigma,s/E} = \{[t]_E \in T_{\Sigma,k/E} \mid E \vdash (\forall \emptyset) t : s\}$; this definition does not depend on the choice of a particular term t by virtue of the* Membership *rule of* MEL *in Definition 1.* □

We write $E \vdash_{ind} \varphi$ to express the fact that the MEL sentence φ is valid in the initial model of the MEL theory (Σ, E). The next proposition considers *implications* between an atomic ground membership and a general MEL sentence, and characterises the validity of this implication in the initial model. We could have considered more general settings, but we have chosen to prove only what our purposes require.

Proposition 1 *Consider a* MEL *sentence $(\forall X)\, \psi = (\forall X)\, \psi_0$ if $\psi_1 \wedge \ldots \wedge \psi_n$, where all the ψ_i, for $i \in [0, n]$, are atomic formulas of a* MEL *theory (Σ, E). Consider also a ground atomic membership $(\forall \emptyset) t : s$ with $t \in T_\Sigma(X)$. Then, the implication $(\forall \emptyset) t : s \Rightarrow (\forall X)\psi$ is a* MEL *sentence of (Σ, E); and $E \vdash_{ind} ((\forall \emptyset) t : s \Rightarrow (\forall X)\psi)$ iff the following logical implication holds: $[E \vdash (\forall \emptyset) t : s \Longrightarrow E \vdash_{ind} (\forall X)\psi]$.* □

The next proposition allows an existential quantifier to move from the right-hand-side of the $\vdash_{ind}$ relation to its left-hand side. For a MEL formula with a free variable x, we denote by $\varphi(t/x)$ the formula obtained by substituting in φ the variable x by the same-kinded term t.

Proposition 2 *Let $(\forall x : [k])(\forall X)\varphi$ be a sentence of a* MEL *theory (Σ, E), for some kind k of the theory, such that $x \notin X$. Then, $E \vdash_{ind} (\forall x : [k])(\forall X)\varphi$ iff $E \vdash_{ind} (\forall X)\varphi(t/x)$ for all ground terms $t \in T_{\Sigma,k}$.* □

2.3 Deduction of Membership Assertions

In the rest of this section we give a technical result that will be used in the core Section 4. We focus on the deduction of *membership assertions*. We shall find it convenient to add a (redundant) rule to the deduction system of MEL:

Definition 3 *Given a* MEL *theory $\mathcal{M} = (\Sigma, E)$, the enriched deduction system of $\mathcal{M}$ consists of the rules given in Definition 1 and, additionally, the following rule*

2'. Membership' : $\frac{(\mu)\ (\forall X) t:s \in E}{E \vdash (\forall X) t:s}$ □

This rule is redundant because it can be deduced from the *Replacement*$_2$ rule using the identity substitution. Hence, deduction in the enriched system is equivalent to deduction in the "normal" one. However, we shall need to distinguish applications of the *Membership'* rule from (other) applications of the *Replacement*$_2$ rule; this is because some results refer to derivations in which the *Replacement*$_2$ rule is applied with some restrictions. In the sequel, when we refer to the deduction system of a MEL theory, we mean the enriched deduction system from Definition 3.

The next definition identifies the sentences in a MEL theory that "define" the membership in a given sort:

Definition 4 *The definition $\mathcal{D}(s)$ of a sort s of a* MEL *theory $\mathcal{M} = (\Sigma, E)$ is the set of assertions $\{(\mu)\ (\forall X)t : s'$ if $C \quad \in E | s' = s\}$. The elements of $\mathcal{D}(s)$ are called the* defining assertions *of the sort s.* □

The following definition focuses on the memberships that can be defined using a membership assertion exactly once. We say that the *Replacement*$_2$ rule is *used with* a given membership μ; this means that, in a given application of the *Replacement*$_2$ rule, μ is the membership occurring in the premise of the rule.

Definition 5 *A membership $(\mu) : (\forall X)t : s$ if $C \ \in \mathcal{D}(s)$ of a* MEL *theory $\mathcal{M} = (\Sigma, E)$* entails *an assertion $(\forall X)t : s$, denoted by $E \vdash_\mu (\forall X)t : s$, if the assertion $(\forall X)t : s$ can be derived by using the deduction rules in Definitions 1 and 3, with the following restrictions:*

- *there is exactly one use of the Replacement$_2$ rule with a membership in $\mathcal{D}(s)$;*
- *the application of the Replacement$_2$ rule from the previous item is used with the membership μ.* □

The above definition will be used in Section 4 when we encode the effect of a single rewrite rule by the effect of a single membership assertion.

The following definition allows to separate from a membership's condition the part that refers to a sort s and the part that does not refer to the sort:

Definition 6 *For $(\mu)(\forall X)t : s$ if $C_\mu \ \in \mathcal{D}(s)$ a defining assertion for sort s, with condition $C_\mu : \bigwedge_{i\in I}(\forall X)(u_i = v_i) \wedge \bigwedge_{j\in J}(\forall X)(w_j : s_j)$, we let $C_{s,\mu} : \bigwedge_{j\in J, s_j = s}(\forall X)(w_j : s_j)$, and $C_{\neg s,\mu} : \bigwedge_{i\in I}(\forall X)(u_i = v_i) \wedge \bigwedge_{j\in J, s_j \neq s}(\forall X)(w_j : s_j)$.* □

Of course, any of the conjunctions $C_{s,\mu}$, $C_{\neg s,\mu}$ (or both) may be empty, in which case we conveniently identify them with the Boolean constant *true*. Then, $C_\mu = C_{s,\mu} \wedge C_{\neg s,\mu}$ for all memberships $(\mu)\ t : s$ *if* $C \quad \in \mathcal{D}(s)$.

Definition 7 *A defining assertion of a sort s: $(\mu)\ \ (\forall X)t : s$ if $C \in \mathcal{D}(s)$ with $C : \bigwedge_{i\in I}(\forall X)(u_i = v_i) \wedge \bigwedge_{j\in J}(\forall X)(w_j : s_j)$ is* inductive *if there exists $j \in J$ such that $s_j = s$.* □

Note that our notion of inductive assertion for a sort does not cover, e.g., mutually inductive definitions; however, it is sufficient for our purposes. We shall often say that the equations $(\forall X)(u_i = v_i)$ and memberships $(\forall X)w_j : s_j$ *occur* in the condition $C : \bigwedge_{i\in I}(\forall X)(u_i = v_i) \wedge \bigwedge_{j\in J}(\forall X)(w_j : s_j)$. Similarly, we say that the sorts s_j (for $j \in J$) *occur* in the condition C.

Definition 8 *A sort s of a* MEL *theory $\mathcal{M} = (\Sigma, E)$ is called* simply inductive *if, for all defining assertions (μ) $(\forall X)t : s$ if $C_{s,\mu} \wedge C_{\neg s,\mu} \in \mathcal{D}(s)$, the conjunct $C_{s,\mu}$ is either* true *or a atomic membership formula of the form $(\forall X)t : s$; and the sort s does not occur in the conditions of sentences in $E \setminus \mathcal{D}(s)$.* □

That is, memberships of the form $(\forall X)t : s$ may only occur in the condition of a defining assertion of the sort s at most once, and they may not occur in the condition of any other equation or membership in E.

For example, consider a theory containing only the definition of the sort `Nat` using the assertions `mb 0 : Nat` and `cmb s(n) : Nat if n : Nat`. Then, `Nat` is simply inductive. But if the theory also contains a definition of the sort `Int` using the assertions `cmb i : Int if i : Nat` and `cmb i : Int if minus(i) : Nat`, then `Nat` is not simply inductive, because it also occurs in the right-hand sides of assertions that define `Int`. However, `Int` is simply inductive.

The main result in this section characterises the structure of derivations for entailments of the form $E \cup \{(\forall Y)t : s\} \vdash_\mu (\forall Y)t' : s$, under some hypotheses on the sort s.

Proposition 3 *Consider a* MEL *theory $\mathcal{M} = (\Sigma, E)$, and let s be simply inductive sort of $\mathcal{M}$ such that all the defining assertions of s in E are inductive. Then, for all $t, t' \in T_\Sigma(X)$: $E \cup \{(\forall X)t : s\} \vdash (\forall X)t' : s$ iff either $E \vdash (\forall X)t = t'$, or there exists a sequence of memberships $\mu_1, \ldots, \mu_n \in E \cap D(s)$ $(n \geq 1)$ and a sequence of terms $t_i \in T_\Sigma(X)$ for $i \in [0, n-1]$ such that $t_0 = t$, $t_n = t'$, and $E \cup \{(\forall X)t_i : s\} \vdash_{\mu_{i+1}} (\forall X)t_{i+1} : s$ for $i \in [0, n-1]$.* □

3 Rewriting Logic

We start this section by a brief presentation of Rewriting Logic (RL) [1] as implemented in the Maude tool [2]. After the basic notions, presented in Section 3.1, we introduce the notion of top-level rewriting in Section 3.2 and the notion of invariant of a rewrite theory in Section 3.3. We also give a technical result, to be used in the next section.

3.1 Basic Notions

A Rewriting Logic theory is a tuple $\mathcal{R} = (K, \Sigma, S, E, R)$ (often abbreviated as (Σ, E, R)), where (Σ, E) is a MEL theory, and R is a set of *rewrite rules*. In this paper we consider rules of the form:

$$(\rho) \quad (\forall X)\, l \to r \quad \textit{if} \quad C \tag{3}$$

where the condition C has the form $\bigwedge_{i\in I}(\forall X)(u_i = v_i) \wedge \bigwedge_{j\in J}(\forall X)(w_j : s_j)$ for some finite sets of indices I and J; that is, like for MEL sentences, only equations and memberships are allowed in the condition[3]. We label rules in R by unique labels and refer to the rules by their labels. A *sequent* is a pair of universally quantified terms of the same kind, denoted by $(\forall X)t \to t'$ with $t, t' \in T_{\Sigma,k}(X)$ for some kind $k \in K$.

Definition 9 *The theory $\mathcal{R} = (\Sigma, E, R)$* entails *the sequent $(\forall X)t \to t'$, denoted by $\mathcal{R} \vdash (\forall X)t \to t'$, if the sequent $(\forall X)t \to t'$ can be obtained by applying the following rules*[4].

1. *Reflexivity:* $\frac{t\in T_\Sigma(X)}{\mathcal{R}\vdash(\forall X)t\to t}$
2. *Transitivity:* $\frac{\mathcal{R}\vdash(\forall X)t_1\to t_2 \quad \mathcal{R}\vdash(\forall X)t_2\to t_3}{\mathcal{R}\vdash(\forall X)t_1\to t_3}$
3. *Equality:* $\frac{E\vdash(\forall X)t=u \quad \mathcal{R}\vdash(\forall X)u\to u' \quad E\vdash(\forall X)u'=t'}{\mathcal{R}\vdash(\forall X)t\to t'}$
4. *Congruence:*
$$\frac{f\in\Sigma_{k_1,\ldots,k_n,k} \quad t_1,\ldots t_n\in T_{\Sigma(X)_{k_i}} \text{ for } i\in[1,n] \quad t'_i\in T_{\Sigma(X)_{k_i}} \quad \mathcal{R}\vdash(\forall X)t_i\to t'_i}{\mathcal{R}\vdash(\forall X)f(t_1,\ldots,t_i,\ldots,t_n)\to f(t_1,\ldots,t'_i,\ldots,t_n)}$$
5. *Replacement:* $\frac{(\rho)\ (\forall X)\ l\to r \text{ if } C\in R \quad \sigma:X\mapsto T_\Sigma(Y) \quad E\vdash(\forall Y)C\sigma}{\mathcal{R}\vdash(\forall Y)l\sigma\to r\sigma}$ □

Intuitively, the *Replacement* rule is the one that actually performs rewriting (with a given rule ρ); the *Congruence* rule delegates rewriting on a subterm; the *Equality* and *Reflexivity* rules ensure consistency with the underlying MEL; and the *Transitivity* rule allows for sequential composition of rewrites.

As an example, consider the Maude specification of Lamport's Bakery algorithm for two processes (Figure 2). In Section 5.3 we shall consider a generalisation of the algorithm to n processes. The module `BAKERY` starts by importing the module `INT` for Integers. Then, six sorts are declared. The sorts `Int?` and `Bool?` are supersorts of the usual integers and Booleans; these supersorts are meant to play the role of *kinds* - they are "kind surrogates" that the ITP tool needs for technical reasons. The four other sorts are `Loc` (for control locations) and `State` for the system's state, together with their "kind surrogates", `Loc?` and `State?`. Next, three control locations (of sort `Loc`) are defined: `Sleep`, `Try`, and `Critical`.

The global state of the system is given by the constructor `<_,_,_,_>` that takes four arguments (the locations of the two processes and their integer *tickets*) and returns a `State?`. The conditional membership associated to the operator ensures that it builds a proper `State` when its arguments are proper locations `Loc` and integers `Int`. The initial state `Init` is defined to be `<Sleep,Sleep,0,0>`, i.e., both processes are sleeping and their tickets have value zero; and the rest of the specification describes the transitions of the processes.

[3] In its most general form [6], rewriting logic also allows for *rewrites in conditions* and *frozen arguments*, which we do not consider here.

[4] In its most general form [6], the Replacement rule allows for so-called *Nested Replacements*, and the Congruence rule allows for parallel rewriting of subterms. As noted in [6], for *reachability*, which is our main concern here, these differences do not matter.

```
mod BAKERY is
protecting INT . --- module importation
sorts Bool? Int? Loc Loc?  State State? .
subsort Int < Int? .
subsort Loc < Loc? .
subsort State < State? .
subsort Bool < Bool? .

ops Sleep Try Critical : -> Loc?  .
mb Sleep : Loc .
mb Try : Loc .
mb Critical : Loc .

vars l1 l2 l' : Loc? .
vars t1 t2 : Int? .

op <_,_,_,_> : Loc? Loc? Int? Int? -> State? [ctor] .
cmb < l1 , l2, t1, t2 > : State
if l1 : Loc /\ l2 : Loc  /\ t1 : Int /\ t2 : Int .

op Init : -> State? .
eq Init = < Sleep, Sleep, 0, 0 > .

ops Trans11 Trans12 Trans13 Trans14
    Trans21 Trans22 Trans23 Trans24 : State? -> State? .

 --- transitions of Process 1
eq Trans11(< Sleep, l2, t1, t2 > ) = < Try , l2, t2 + 1, t2 > .
rl [Trans11] : < Sleep, l2, t1, t2 > => Trans11(< Sleep, l2, t1, t2 >) .

eq Trans12(< Try, l2, t1, 0 > ) = < Critical, l2, t1, 0 >  .
rl [Trans12] : < Try, l2, t1, 0 > => Trans12(< Try, l2, t1, 0 >) .

eq Trans13(< Try, l2, t1, t2 > ) = < Critical, l2, t1, t2 >  .
crl [Trans13] : < Try, l2, t1, t2 > => Trans13(< Try, l2, t1, t2 >) if  t1 < t2 .

eq Trans14(< Critical, l2, t1, t2 > ) = < Sleep, l2, 0, t2 >  .
rl [Trans14] : < Critical, l2, t1, t2 > => Trans14(< Critical, l2, t1, t2 >) .

 ---  transitions of Process 2
eq Trans21(< l1, Sleep, t1, t2 > ) = < l1, Try, t1, t1 + 1 > .
rl [Trans21] : < l1, Sleep, t1, t2 >  => Trans21(< l1, Sleep, t1, t2 >)  .

eq Trans22(< l1, Try, 0, t2 > ) = < l1, Critical, 0, t2 > .
rl [Trans22] : < l1, Try, 0, t2 > => Trans22(< l1, Try, 0, t2 > ) .

eq Trans23(< l1, Try, t1, t2 > ) = < l1, Critical, t1, t2 > .
crl [Trans23] : < l1, Try, t1, t2 > => Trans23(< l1, Try, t1, t2 >) if t2 < t1 .

eq Trans24(< l1, Critical, t1, t2 > ) = < l1, Sleep, t1, 0 > .
rl [Trans24] : < l1, Critical, t1, t2 > => Trans24(< l1, Critical, t1, t2 > ) .
                                                                                Irisa
endm
```

Figure 2: Maude specification of a 2-process Bakery algorithm.

For technical reasons related to the executability of specifications (more about this in Section 5) we have chosen to split each transition into two parts. The first part defines an operator called $\texttt{trans}_{n,m}$, for $n, m \in [1, 4]$, using an equation, e.g., `eq Trans13(< Try, l2, t1, t2 > ) = < Critical, l2, t1, t2 >`. The second part is a rewrite rule, that rewrites certain states to the corresponding application of $\texttt{trans}_{n,m}$ applied to those states, i.e., `crl [Trans13]:< Try, l2, t1, t2 > => Trans13(< Try, l2, t1, t2 >) if t1 < t2`. The rules can be conditional (`crl` keyword), like the above rule, which says that Process 1 can only enter the critical section if its ticket `t1` is smaller than the ticket `t2` of Process 2; or they can be unconditional, as, e.g., the other condition under which Process 1 may enter the critical section - when Process 2 is not interested, i.e., when `t2 = 0`: `< Try, l2, t1, 0 > => Trans12(< Try, l2, t1, 0 >)` where `Trans12(< Try, l2, t1, 0 >) = < Critical, l2, t1,0 >`.

The remaining transitions of Process 1 are now described: in `Trans11`, the process goes from `Sleep` to `Critical` and sets its ticket `t1` to `t2+1`; and in `Trans14`, the process goes back to `Sleep`, setting its ticket to zero.

The transitions of Process 2 are symmetrical to those of Process 1. The system is infinite state, because even though each ticket is repeatedly reset to zero, the maximum of the two tickets may grow beyond any bound.

Let us now consider the problem of *verifying* that the Bakery algorithm satisfies mutual exclusion to `Critical`. One way to proceed is to use Maude's *search* command, looking for a state `s` of the form `<Critical, Critical, t1, t2>` reachable from the initial state `Init`. Of course, the system is infinite-state and *search* may only explore finitely many reachable states (i.e., an *under*-approximation), hence, there is no guarantee that no erroneous states are reachable even if *search* does not report any.

Another approach is to use so-called *equational abstractions* [8], which collapses the infinite-state system into a finite-state one that *over*-approximates the set of reachable states. The approach is sound, in the sense that if no state violating mutual exclusion is found in the finite-state abstraction then there is no violating state in the initial infinite-state system either (there are no false negatives). However, the approach does not guarantee that violating states in the abstraction do correspond to actually violating states in the original system (false positives), and getting rid of false positivs requires the user to *refine* the abstraction. Automating such abstraction/refinement verification techniques in the context of rewriting logic, in the spirit of *counterexample*-guided abstraction refinement [14] is a matter for future work. Another approach is to perform *interactive theorem proving* as proposed here.

3.2 Top-level rewriting

Top-level rewriting is defined in [10], where it is shown shown that top-level rewriting is enough for describing the dynamics of systems in many practical cases. We give some intermediary results about top-level rewriting, to prepare for the main results in Section 4.

Definition 10 *The theory* $\mathcal{R}$ entails *the sequent* $(\forall X)t \rightarrow t'$ at top level, *denoted by* $\mathcal{R} \vdash (\forall X)t \hookrightarrow t'$, *if the sequent* $(\forall X)t \rightarrow t'$ *can be derived by applying the rules given in Definition 9 except* Congruence. □

As an example, consider the 2-process Bakery algorithm from Figure 2. We show that rewriting using the rules given in Definition 9 actually reduces to top-level rewriting (Definition 10). For this, note that all the rewrite rules operate on the kind `[State]`, and that this kind (represented in our algorithm by the "kind-surrogate" `State?` required by the ITP tool for technical reasons) is defined by the operators `<_,_,_,_> : Loc? Loc? Int? Int? -> State?` and `Init : -> State?`. Note that the operators are *not recursive* that is, a term of kind `[State]` cannot have subterms that also have kind `[State]`. Hence, the rewrite rules cannot apply to subterms, that is, all the rewriting is performed at top level. This observation can be generalised as follows.

Observation 1 *Let* $\mathcal{R} = (K, \Sigma, S, E, R)$ *be a rewrite theory and a kind* $k \in K$. *Assume that the none of the operators* $f : k_1, \ldots, k_n \mapsto k \quad \in \Sigma$ *is recursive, i.e.,* $k_i \neq k$ *for all* $i \in [1, n]$, *and that all the rewrite rules in* R *operate on the kind* k, *i.e., for all* $(\forall l \rightarrow r \text{ if } C \quad \in R$, *we have* $l, r \in T_{\Sigma,k}(X)$. *Then, for all* $t, t' \in T_{\Sigma,k}(X)$: $\mathcal{R} \vdash (\forall X)t \rightarrow t'$ *iff* $\mathcal{R} \vdash (\forall X)t \hookrightarrow t'$. □

Proof (sketch). The ($\Leftarrow$) implication is trivial. ($\Rightarrow$) is established by noting that *Congruence* is the only rule that might make a difference between the $\rightarrow$ and $\hookrightarrow$ entailments, (cf. Definitions 9, 10), but that *Congruence* cannot be applied in deriving the $\rightarrow$ entailment since its premise $\mathcal{R} \vdash (\forall X)t_i \rightarrow t_i'$ is not satisfied (as $t_i, t_i' \in T_{\Sigma,k_i}(X)$ and $k_i \neq k$). □

Observation 1 gives us sufficient conditions under which the rewriting reduces to top-level rewriting. The next result shows us how to *transform* a rewrite theory $\mathcal{R}$ into another rewrite theory $\{\mathcal{R}\}$, called the *encapsulation* of $\mathcal{R}$, such that rewriting at top level in $\mathcal{R}$ is equivalent to rewriting in $\{\mathcal{R}\}$. This is useful if we want to simulate top-level rewriting by using an existing (general) rewriting mechanism such as that of Maude.

Definition 11 *Let* $\mathcal{R} = (K, \Sigma, S, E, R)$ *a rewrite theory and a kind* $k \in K$ *such that all the rewrite rules in* R *operate on the kind* k. *We define the encapsulation* $\{\mathcal{R}\} = (K_{\{\mathcal{R}\}}, \Sigma_{\{\mathcal{R}\}}, S_{\{\mathcal{R}\}}, E_{\{\mathcal{R}\}}, R_{\{\mathcal{R}\}})$ *to be the rewrite theory defined as follows:*

- $K_{\{\mathcal{R}\}} = K \cup \{k'\}$, *where* $k' \notin K$ *is a new kind,*
- $\Sigma_{\{\mathcal{R}\}} = \Sigma \cup \{\{\_\} : k \mapsto k'\}$,
- $S_{\{\mathcal{R}\}} = S$,
- $E_{\{\mathcal{R}\}} = E$,
- $R_{\{\mathcal{R}\}} = \{\{l\} \rightarrow \{r\} \text{ if } C \mid l \rightarrow r \text{ if } C \quad \in R\}$. □

That is, the unary operator $\{\_\}$ just "encapsulates" all the rewrite rules, such that all rewriting now takes place on terms of the (new) kind k'.

Observation 2 *Let $\mathcal{R} = (K, \Sigma, S, E, R)$ be a rewrite theory, a kind $k \in K$ such that all the rewrite rules in R operate on the kind k, and $t, t' \in T_{\Sigma,k}(X)$. Then, $\mathcal{R} \vdash (\forall X)t \hookrightarrow t'$ iff $\{\mathcal{R}\} \vdash (\forall X)\{t\} \rightarrow \{t'\}$.* □

Proof (sketch). We first note that the theory $\{\mathcal{R}\}$ and the kind k' satisfy all the conditions of Lemma 1, that is, none of the operators with co-domain k' is recursive (there is only one such operator $\{\_\}$, which clearly is not recursive), and all the rewrite rules operate on the kind k'. Hence, we can apply Lemma 1 and obtain that $\{\mathcal{R}\} \vdash (\forall X)\{t\} \rightarrow \{t'\}$ iff $\{\mathcal{R}\} \vdash (\forall X)\{t\} \hookrightarrow \{t'\}$. It remains to prove that $\mathcal{R} \vdash (\forall X)t \hookrightarrow t'$ iff $\{\mathcal{R}\} \vdash (\forall X)\{t\} \hookrightarrow \{t'\}$, which is easily done by induction on the derivations. □

We continue this section with the definition of *top-level entailment using a single rewrite rule*. Note the analogies with Definition 5 from Section 2, of entailment using a single membership - this is no coincidence - in the next section we encode rewrites using memberships.

Definition 12 *A rule ρ of a* RL *theory $\mathcal{R} = (\Sigma, E, R)$* entails *a sequent $(\forall X)t \rightarrow t'$* at top level, *denoted by $\mathcal{R} \vdash_\rho (\forall X)t \hookrightarrow t'$, if $(\forall X)t \rightarrow t'$ can be derived at top-level, and*

- *the Replacement rule is used exactly once*
- *the application of the Replacement rule from the previous item is used with the rule ρ.* □

The main result in this section characterises the top-level rewriting as a sequence of one-step top-level rewrites. Note the analogy with Proposition 3 from Section 2.

Proposition 4 *Consider a* RL *theory $\mathcal{R} = (\Sigma, E, R)$. Then, for all $t, t' \in T_\Sigma(X)$: $\mathcal{R} \vdash (\forall X)t \hookrightarrow t'$ iff either $E \vdash (\forall X)t = t'$, or there exists a sequence of rules $\rho_1, \ldots, \rho_n \in R$ ($n \geq 1$) and terms $t_i \in T_\Sigma(X)$, $i \in [0, n-1]$ such that $t_0 = t$, $t_n = t'$, and $\mathcal{R} \vdash_{\rho_i} (\forall X)t_i \hookrightarrow t_{i+1}$ for $i \in [0, n-1]$.* □

3.3 Invariants of Rewrite Theories

Intuitively, a predicate φ over the states of a dynamic system $\mathcal{R}$ is an invariant property (or, simply, an invariant), denoted by $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$, if $\varphi(t) = \mathit{true}$ for all the states t that are reachable from the initial state t_0 of the system. We now specialise these notions when the system $\mathcal{R} = (K, \Sigma, S, E, R)$ is a RL theory. We have to settle to following details:

1. what are the *states* and the *dynamics* when $\mathcal{R}$ is a rewrite theory?
2. what is the precise *syntax* of the state predicates φ?
3. what is the precise *semantics* of the statement $\varphi(t) = \mathit{true}$?
4. what is the precise *semantics* of the statement $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$?

For the first question: in our setting, states shall be ground terms of a certain kind, say, $[State] \in K$ and the dynamics shall be defined by top-level rewriting in $\mathcal{R}$, starting from an initial term t_0 (not necessarily ground) of kind $[State]$. Having non-ground initial terms allows for several, even infinitely many initial states, useful for describing, e.g., systems with a parametric number of processes, such as the n-process Bakery algorithm in Section 5.3.

For the second question: state predicates φ have to refer in some way to the kind $[State]$. A reasonable definition is that state predicates are Horn sentences, built on atomic formulas such that at least one atom in the sentence is

- either a predicate that has at least one argument of kind $[State]$
- or a membership to the sort $State$.

Indeed, if none of the atoms of the sentence φ is of either forms above, then the formula does not refer to the kind $[State]$ at all, and then it does not make sense to call φ a state predicate. On the other hand, it is not necessary that *all* the atoms of φ refer to the kind $[State]$; some atoms may only refer to other, auxiliary variables of other kinds, as we shall see it useful in Section 5.3. Also, restriction to Horn sentences is not arbitrary; even if the ITP theorem prover allows for more general first-order formulas, working with such formulas in ITP is somewhat complicated because such formulas are not known to Maude.

For the third question: the semantics of $\varphi(t) = true$ shall be that induced by the initial model of the MEL sub-theory (K, Σ, S, E) of $\mathcal{R}$And for the fourth question: we shall say $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$ whenever for all ground terms $t \in T_\Sigma$: if $\mathcal{R} \vdash (\forall X) t_0 \hookrightarrow t$ then $E \vdash_{ind} \varphi(t) = true$. This formalises the intuitive idea that invariants hold in all reachable states.

Definition 13 *Given a rewrite theory $\mathcal{R} = (K, \Sigma, S, E, R)$, we say that $\mathcal{R}$ is* admissible *if*

- *there exists a kind $[State] \in K$ and a sort $State \in S_{[State]}$;*
- *there exists a kind $[Bool] \in K$ and a sort $Bool \in S_{[Bool]}$, and two constants $true : Bool$ and $false : Bool$ such that for all $b : Bool$, either $E \vdash b = true$ or $E \vdash b = false$.* □

The last constraint ensures that Booleans are defined and correctly interpreted in (Σ, E).

Definition 14 *Given an admissible rewrite theory $\mathcal{R} = (K, \Sigma, S, E, R)$,* a state predicate *of $\mathcal{R}$ is a universally quantified Horn sentence of the form*

$$(\forall x : [State], \forall X)\, \varphi_0 \text{ if } \varphi_1 \wedge \ldots \wedge \varphi_n$$

where $x \notin X$, and, for $i \in [0, n]$, φ_i is either of

- *Type 1: a predicate with signature of the form $k_1 \times \ldots \times k_m \mapsto [Bool]$ for some nonempty product of kinds $\prod_{j=1}^{m} k_j \in K^m$ $(m \geq 1)$, or*
- *Type 2: a membership $t' : s$, for some term $t' \in T_\Sigma(\{x\} \cup X)$,*

and such that there exists at least one atomic formula φ_i either of Type 2 with $s = State$, or of Type 1 with $k_j = [State]$ for some $j \in [1, m]$. □

Note that, for all $t \in T_\Sigma$, $(\forall X)\varphi[t/x]$, denoted hereafter by $(\forall X)\varphi(t)$, is a MEL sentence in the signature Σ. In particular, its validity in the initial model of (Σ, E) is defined. The following definition formalises the notion of invariant.

Definition 15 ($\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$) *For an admissible* RL *theory $\mathcal{R} = (\Sigma, E, R)$, a term $t_0 \in T_{\Sigma,[State]}(X)$, and a state predicate φ of $\mathcal{R}$, we say that φ is an invariant of $\langle \mathcal{R}, t_0 \rangle$, denoted by $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$, if, for all ground terms $t \in T_{\Sigma,[State]}$, if $\mathcal{R} \vdash (\forall X) t_0 \hookrightarrow t$ then $E \vdash_{ind} (\forall X)\varphi(t)$.* □

Note that the restriction to terms $t \in T_{\Sigma,[State]}$ is not mandatory - we could have let $t \in T_\Sigma$ instead - because all terms reachable from a term of kind $[State]$ also have kind $[State]$, a fact easily proved by induction.

We close this section with an example. Consider again the 2-process Bakery algorithm depicted in Figure 2, and the state predicate `mutEx : State? -> Bool?`[5] defined in Figure 3, which uses the simpler predicate `isCritical`. A location satisfies `isCritical` if and only if it *is* `Critical`, and the predicate `mutex` is satisfied by a term of sort `State?` (which plays the role of the kind $[State]$ in the above formalisation, cf. the discussion around the 2-process Bakery algorithm) iff its two locations are `Critical`.

Can we establish $\langle$`BAKERY, Init`$\rangle \vdash \Box$`mutEx` directly using Definition 15? Unfortunately, no, because the definition requires us to know all the terms reachable from the term `Initial` and there are infinitely many such terms. We shall see in the next section an alternative definition, equivalent to Definition 15, which is well adapted to interactive theorem proving.

4 Encoding RL Theories in MEL Theories

This section describes our theorem-proving based approach for verifying invariance properties of rewrite theories. The section has two parts.

The first part, Section 4.1, describes an automatic translation that takes a RL theory $\mathcal{R}$ and a term t_0, and generates a MEL theory $\mathcal{M}(\mathcal{R}, t_0)$, which enriches $\mathcal{R}$ with a sort called *Reachable* and with memberships defining it. We prove that "being reachable in $\mathcal{R}$ from t_0 by top-level rewriting" and "having sort *Reachable* in $\mathcal{M}(\mathcal{R}, t_0)$" are equivalent statements.

Motivated by this equivalence, in Section 4.2 we define an alternative definition of invariant φ of a RL theory $\mathcal{R}$ as follows: $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi$ *if $\varphi(t)$ = true in the initial model of $\mathcal{M}(\mathcal{R}, t_0)$, for all terms t of sort Reachable.* We prove the equivalence of $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi$ with $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$ given by Definition 15. Since the definition of $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi$ uses only MEL concepts, it can be used for proving invariants using the ITP tool.

[5] The signature of our state predicate is the simplest possible, i.e., `State?->Bool?`. In Section 5.3 we show more general state predicates obeying to Definition 14.

```
var s : State? .
vars l1 l2 : Loc?
vars t1 t2 : Int?

op isCritical : Loc? -> Bool? . --- is a location Critical?
eq isCritical(Sleep) = false .
eq isCritical(Try) = false .
eq isCritical(Critical) = true .

op mutEx : State? -> Bool? .    --- defines mutual exclusion
cmb mutEx(s) : Bool if s : State .
--- mutex is violated  when both processes are in critical section
eq mutEx(< Critical, Critical, t1, t2 > ) = false .
--- otherwise, mutex is satisfied
ceq mutEx(< l1, l2, t1, t2 > ) = true if  isCritical(l1) = false .
ceq mutEx(< l1, l2, t1, t2 > ) = true if  isCritical(l2) = false .
```

Figure 3: Defining Mutual Exclusion.

4.1 Encoding Reachability

We first define the encoding of a rewrite rule by a conditional membership.

Definition 16 *Given a rewrite rule* $\rho : \quad (\forall X)\ l \to r\ \textit{if}\ \ C$, *we denote by* $\mu(\rho)$ *the membership* $\mu(\rho) : \quad (\forall X)\ r : \textit{Reachable if}\ \ l : \textit{Reachable} \wedge C$. □

Here, the sort *Reachable* is a new sort of the MEL theory $\mathcal{M}(\mathcal{R})$, defined below, which "encodes" the reachability relation of a RL theory $\mathcal{R}$. Intuitively, the membership $\mu(\rho)$ "reverses" the rule ρ, expressing the idea that the right-hand side of ρ is reachable whenever its left-hand side is reachable.

Definition 17 *Given an* RL *theory* $\mathcal{R} = (K, \Sigma, S, E, R)$, *we denote by* $\mathcal{M}(\mathcal{R})$ *the* MEL *theory* $(K_{\mathcal{M}(\mathcal{R})}, \Sigma_{\mathcal{M}(\mathcal{R})}, S_{\mathcal{M}(\mathcal{R})}, E_{\mathcal{M}(\mathcal{R})})$ *built as follows:*

- $K_{\mathcal{M}(\mathcal{R})} = K \cup \{[\textit{Reachable}]\}$, *where Reachable is a new sort, and* $[\textit{Reachable}]$ *denotes the corresponding kind*
- $\Sigma_{\mathcal{M}(\mathcal{R})} = \Sigma$
- $S_{\mathcal{M}(\mathcal{R})} = S \cup S_{\textit{Reachable}}$, *where* $S_{\textit{Reachable}} = \{\{\textit{Reachable}\}\}$
- $E_{\mathcal{M}(\mathcal{R})} = E \cup \{\mu(\rho) | \rho \in R\}$, *where* $\mu(\rho)$ *is given by Definition 16.* □

As an example, consider the 2-process RL theory `BAKERY` depicted in Fig. 2. The MEL theory $\mathcal{M}$(`BAKERY`) consists of the MEL part of the `BAKERY` module, enriched with the sort declaration and memberships shown in Fig. 4.

```
sort Reachable .
cmb Trans11(< Sleep, l2, t1, t2 >): Reachable
  if < Sleep, l2, t1, t2 >: Reachable .
cmb  Trans12(< Try, l2, t1, 0 >): Reachable
  if < Try, l2, t1, 0 >: Reachable .
cmb Trans13(< Try, l2, t1, t2 >): Reachable
  if < Try, l2, t1, t2 >: Reachable /\ t1 < t2 .
cmb Trans14(< Critical, l2, t1, t2 >): Reachable
  if < Critical, l2, t1, t2 >: Reachable .
cmb Trans21(< l1, Sleep, t1, t2 >): Reachable
  if < l1, Sleep, t1, t2 > : ReachableState .
cmb Trans22(< l1, Try, 0, t2 > ): Reachable
  if < l1, Try, 0, t2 > : Reachable .
cmb Trans23(< l1, Try, t1, t2 >): Reachable
  if < l1, Try, t1, t2 > : ReachableState /\ t2 < t1 .
cmb Trans24(< l1, Critical, t1, t2 >): Reachable
  if < l1, Critical, t1, t2 >: Reachable .
```

Figure 4: Memberships defining the sort `Reachable`.

There is one membership per rewrite rule. Later we shall see that one last membership is needed to express the fact that the initial state is reachable.

We note that, in Definition 17, the sort *Reachable* is simply inductive (Definition 8) and that all defining assertions of *Reachable* in $E_{\mathcal{M}(\mathcal{R})}$ are inductive (Definition 7). The next proposition states an equivalence between inference using one membership $\mu(\rho)$ and derivation using one rule ρ:

Proposition 5 *With the notations of Definition 17, for all* $t, t' \in T_\Sigma(Y)$: $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : \mathit{Reachable}\} \vdash_{\mu(\rho)} (\forall Y)t' : \mathit{Reachable}$ iff $\mathcal{R} \vdash_\rho (\forall Y)t \hookrightarrow t'$. □

The following theorem establishes the equivalence between membership in the *Reachable* sort and reachability. We shall be using the MEL theory $\mathcal{M}(\mathcal{R}, t)$, defined as follows: $\mathcal{M}(\mathcal{R}, t)$ has the same kinds, signature, and sorts as $\mathcal{M}(\mathcal{R})$, and its set of sentences is that of $\mathcal{M}(\mathcal{R})$ plus the membership $(\forall X)t : \mathit{Reachable}$. For any entailement e, we say $\mathcal{M}(\mathcal{R}, t) \vdash e$ iff $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall X)t : \mathit{Reachable}\} \vdash e$.

Theorem 1 *With the above notations, for all* $t, t' \in T_\Sigma(X)$: $\mathcal{M}(\mathcal{R}, t) \vdash (\forall X)t' : \mathit{Reachable}$ iff $\mathcal{R} \vdash (\forall X)t \hookrightarrow t'$. □

Proof. $\mathcal{M}(\mathcal{R}, t) \vdash (\forall X)t' : \mathit{Reachable}$ means $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall X)t : \mathit{Reachable}\} \vdash (\forall X)t' : \mathit{Reachable}$. We prove the equivalence

$$E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall X)t : \mathit{Reachable}\} \vdash (\forall X)t' : \mathit{Reachable} \textit{ iff } \mathcal{R} \vdash (\forall X)t \hookrightarrow t'.$$

($\Rightarrow$) By Proposition 3, $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall X)t : \mathit{Reachable}\} \vdash (\forall X)t' : \mathit{Reachable}$ iff either $E \vdash (\forall X)t = t'$ (in which case the conclusion follows by the *Reflexivity* and *Equality* rules of

RL), or there exist $n \geq 1$ memberships in $E_{\mathcal{M}(\mathcal{R})} \cap \mathcal{D}(\mathit{Reachable})$ and a sequence of terms $t_i \in T_{\Sigma_{\mathcal{M}(\mathcal{R})}}(X)$, for $i \in [0, n-1]$ such that $t_0 = t$, $t_n = t'$, and $E \cup \{(\forall X) t_i : \mathit{Reachable}\} \vdash_{\mu_{i+1}} (\forall X) t_{i+1} : \mathit{Reachable}$ for $i \in [0, n-1]$. Now, by construction, all memberships in $E_{\mathcal{M}(\mathcal{R})} \cap \mathcal{D}(\mathit{Reachable})$ are obtained from rules in R by the operation $\mu()$ from Definition 16, i.e., there exist $\rho_i \in R$, for $i \in [0, n-1]$, such that $\mu_i = \mu(\rho_i)$. Using n times Proposition 5, we then obtain $\mathcal{R} \vdash_{\rho_i} (\forall X) t_i \hookrightarrow t_{i+1}$ for $i \in [0, n-1]$, and the conclusion follows by Proposition 4.

($\Leftarrow$) By Proposition 4, $\mathcal{R} \vdash (\forall X) t \hookrightarrow t'$ iff either $E \vdash (\forall X) t = t'$ (in which case the conclusion follows by the *Membership* rule of MEL), or there exist $n \geq 1$ rules in R and a sequence of terms $t_i \in T_\Sigma(X) = T_{\Sigma_{\mathcal{M}(\mathcal{R})}}(X)$, for $i \in [0, n-1]$ such that $t_0 = t$, $t_n = t'$, and $\mathcal{R} \vdash_{\rho_i} (\forall X) t_i \hookrightarrow t_{i+1}$ for $i \in [0, n-1]$. Using n times Proposition 5, we then obtain $E \cup \{(\forall X) t_i : \mathit{Reachable}\} \vdash_{\mu_{i+1}} (\forall X) t_{i+1} : \mathit{Reachable}$ for $i \in [0, n-1]$, and the conclusion follows by Proposition 3. □

To conclude this section, we note that the theory $\mathcal{M}(\texttt{BAKERY}, \texttt{Initial})$ enriches the theory $\mathcal{M}(\texttt{BAKERY})$ in Figure 4 with the single membership `mb Init : Reachable` to express the reachability of the initial state.

4.2 Encoding Invariance Proofs

We have established that membership in the *Reachable* sort and top-level reachability are equivalent statements. Motivated by this equivalence, we provide an alternative to Definition 15 of invariant of a rewrite system $\mathcal{R}$, and we prove that the two definitions are equivalent. In the next section we show how the second definition can be be used with ITP.

Definition 18 *For an admissible* RL *theory $\mathcal{R}$, a term $t_0 \in T_{\Sigma,[State]}(X)$, and a state predicate φ of $\mathcal{R}$, we denote by $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi$ the statement $\mathcal{M}(R, t_0) \vdash_{ind} (\forall x : [State])(x : \mathit{Reachable} \Rightarrow (\forall X)\varphi)$.* □

The next proposition is required for proving the equivalence of Definitions 15 and 18. It says that the theory $\mathcal{M}(\mathcal{R}, t_0)$ has the same power for proving atomic state predicates as the underlying MEL theory of the rewrite theory $\mathcal{R}$.

Proposition 6 *Let $\mathcal{R} = (\Sigma, E, R)$ be an admissible* RL *theory and e an atomic state predicate of $\mathcal{R}$, i.e., either a membership $t : s$ or an equation $t = t'$, with $t, t' \in T_\Sigma(X)$. Then, $E \vdash (\forall X) e$ iff $\mathcal{M}(\mathcal{R}, t_0) \vdash (\forall X) e$.* □

A consequence of the above proposition is that, except for the *Reachable* sort, the theories (Σ, E) and $\mathcal{M}(\mathcal{R}, t_0)$ have the same initial model. Hence, for any state predicate φ and term $t \in T_{\Sigma,[State]}$, the MEL sentence $(\forall X)\varphi(t)$ is valid in the initial model of $\mathcal{M}(\mathcal{R}, t_0)$ iff it is valid in the initial model of (Σ, E), as (cf. Def. 14) φ does not refer to the sort *Reachable*.

Corollary 1 *With the notations and under the conditions of Proposition 6, for all state predicates φ, $\mathcal{M}(\mathcal{R}, t_0) \vdash_{ind} (\forall X)\varphi(t)$ iff $E \vdash_{ind} (\forall X)\varphi(t)$.* □

Theorem 2 *Let $\mathcal{R}$ be an admissible* RL *theory, φ a state predicate of $\mathcal{R}$, and $t_0 \in T_{\Sigma,[State]}(X)$. Then, $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi$ iff $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$.* □

Proof. By Definition 18, the statement $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi$ means

$$\mathcal{M}(R, t_0) \vdash_{ind} (\forall x : [State])(x : Reachable \Rightarrow (\forall X)\varphi) \tag{4}$$

which, by Proposition 2, is equivalent to

$$\forall t \in T_{\Sigma,[State]}.\, \mathcal{M}(R, t_0) \vdash_{ind} ((\forall \emptyset)t : Reachable \Rightarrow (\forall X)\varphi(t)) \tag{5}$$

which, by Proposition 1, is equivalent to

$$\forall t \in T_{\Sigma,[State]}.\, \mathcal{M}(R, t_0) \vdash (\forall \emptyset)t : Reachable \Rightarrow \mathcal{M}(R, t_0) \vdash_{ind} (\forall X)\varphi(t) \tag{6}$$

Now, using Theorem 1, $\mathcal{M}(\mathcal{R}, t_0) \vdash (\forall \emptyset)t : Reachable$ is equivalent to $\mathcal{R} \vdash (\forall X)t_0 \hookrightarrow t$; and, by Corollary 1, $\mathcal{M}(\mathcal{R}, t_0) \vdash_{ind} (\forall X)\varphi(t)$ iff $E \vdash_{ind} (\forall X)\varphi(t)$. Hence, the implication (6) yields equivalently

$$\forall t \in T_{\Sigma,[State]}.\, \mathcal{R} \vdash (\forall X)t_0 \hookrightarrow t \Rightarrow E \vdash_{ind} (\forall X)\varphi(t) \tag{7}$$

which, by Definition 15, is $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$, and the proof is done. □

5 Application to Bakery Algorithms

In this Section we illustrate the approach on Bakery algorithms. We start with the 2-process version of the algorithm. We then discuss some executability issues, which show that our approach can deal with systems that are not directly executable by Maude. These issues are illustrated in the third part of the section, when we describe the verification of the n-process Bakery algorithm.

We first give some general guidelines for using the approach. Typically, one starts with induction and automatic rewriting and decision procedures to try to prove $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi$ automatically. If this does not succeed, the user examines the subgoals left unproved and their ITP *context*, and is usually able to come up with *auxiliary lemmas* $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi'$ such that: (recursively) all goals $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi'$ are proved with ITP; and the results are used to close pending subgoals in the proof of $\langle \mathcal{R}, t_0 \rangle \vdash_{ind} \Box\varphi$ and to progress in the proof.

5.1 Verifying the 2-Process Bakery

We illustrate the approach on the 2-processes Bakery algorithm (cf. Figure 2). The state predicate to be proved invariant is `mutEx` (cf. Figure 3).

After loading and initializing ITP in Maude, the user declares her `goal` and the Maude functional module to which the goal refers. Here, the goal is called `mutex` and the module is `BAKERY`. The goal says that for all states `s` of kind `State?`, if their sort is `Reachable` then `mutEx(s)` holds `true` (here, `A` denotes universal quantification). Note that the ITP input requires a generous amount of parentheses. Then, ITP responds with the following output:

```
(goal mutex : BAKERY  |-
  (A{s:State?}
      (((s:State?): Reachable)
         =>
       ((mutEx(s:State?)) = (true)))) .)
```

Figure 5: Mutual exclusion goal in ITP.

```
==================================
label-sel: mutex@0
==================================
A{s:State?}((s:State? : Reachable)==>(mutEx(s:State?)= true))
++++++++++++++++++++++++++++++++++
```

Figure 6: Output of ITP for the main goal.

```
==================================
label-sel: mutex@1.0
==================================
mutEx(Init)= true
```

Figure 7: First ITP subgoal generated by the induction command.

The goal was given the label `mutex@0`, and this is the currently *selected* goal. We can apply the proof-by-induction command `(ind on s:State?)` to perform induction on the state `s` of inductively defined sort `Reachable`. Then, ITP generates nine subgoals: one for the base case, when `s=Init` (cf. Fig. 7) and eight subgoals corresponding to the eight transitions of the algorithm.

The base case is automatically proved by ITP using the `(auto)` command, which invokes automatic rewriting and decision procedures for arithmetic.

The strategy for proving the remaining subgoals also starts with the `(auto)` command. However, the command does not always succeed by itself; the second subgoal, after application of `(auto)`, gives the output in Fig. 8.

```
==================================
label-sel: mutex@2.0
==================================
mutEx(< Critical,V0#0*Loc?,V0#1*Int?,0 >)= true
```

Figure 8: Second ITP subgoal, after application of `auto`.

In Figure 8, `V0#0*Loc?` and `V0#1*Int?` are arbitrary constants of kinds `Loc?` (locations) and `Int?` (integers), respectively. The subgoal says that *in all reachable states* where the first process is in the `Critical` location, and the second process has ticket zero, mutual exclusion holds. Two questions arise:

- *How was the `mutex@2.0` subgoal generated by* ITP*?* Remember that we asked ITP to perform an induction on the variable `s`, whose sort is declared in the premise of the `mutex` goal to be *Reachable* (cf. Figure 5). The sort *Reachable* was defined using membership assertions in Figure 4, Page 19. It turns out that the `mutex@2.0` subgoal was generated by ITP from the following conditional membership

  ```
  Trans12(< Try, l2, t1, 0 >):  Reachable if < Try, l2, t1, 0 >:  Reachable
  ```

  Indeed, remember the definition of `Trans12` from Figure 2, Page 12:

  ```
  Trans12(< Try, l2, t1, 0 > ) = < Critical, l2, t1, 0 >
  ```

  By combining the two above definitions we obtain equivalently

  ```
  < Critical, l2, t1, 0 >:  Reachable if < Try, l2, t1, 0 >:  Reachable
  ```

  Since we have to prove that `mutEx` holds in all reachable states, we have to prove that it holds in states of the form `< Critical, l2, t1, 0 >`.

  The variables `l2` and `t1` were just replaced by ITP by the constants `V0#0*Loc?` and `V0#1*Int?`, respectively (cf. Figure 8).

- *How can we prove the `mutex@2.0` subgoal?* Apparently, the subgoal is not provable: if `V0#0*Loc? = Critical` mutual exclusion is violated. However, remember that invariants only need to hold on *reachable* states. Hence, one way to prove the subgoal is to prove that, in all reachable states, whenever both processes are in their `Critical` locations, the ticket of the *second* process *cannot be* equal to zero. By examining the protocol, we realise that we shall only need the hypothesis that the *second* process is in the `Critical` location. Here, the user's inspiration and knowledge of the protocol was used. Hence, we proceed as follows: we state and prove the following ITP *auxiliary lemma*:

  ```
  (goal lem-crit2 : BAKERY  |-
    (A{s:State?}
       ((((s:State?): ReachableState)
          & ((isCritical(loc2(s:State?))) = (true)))
         => ((tic2(s:State?) > 0)=(true)))) .)
  ```

  where the operators `loc2` and `tic2` return a state's second location and second ticket, respectively. Proving the `mutex2@0` subgoal now amounts to proving `lem-crit2`. This is performed also by induction on `s` and by using automatic rewriting and decision

procedures; and when this is not enough, the user states auxiliary lemmas that, when proved, allow ITP to complete the proofs of higher-level subgoals or lemmas. Now, proving `lem-crit2` requires another auxiliary lemma: `lem-try2`, shown below, which says `tic2(s) > 0` holds also when the second process is in `Try`.

```
(goal lem-try2 : BAKERY  |-
  (A{s:State?}
     ((((s:State?): ReachableState)
        & ((isTry(loc2(s:State?))) = (true)))
       => ((tic2(s:State?) > 0)=(true)))) .)
```

When proving the auxiliary subgoal `lem-try2` using induction, rewriting, and decision procedures, the proof now succeeds. The reason is that the subgoal is *inductive*: that is, its truth is preserved by all the transitions of our algorithm. More precisely, for all transitions $\texttt{Trans}_{i,j}$, for $i, y \in \{1, 4\}$, the hypothesis that the subgoal is *true* before the transition is fired is enough to establish that the subgoal is still *true* after the transition is fired. Such inductive subgoals are "leaves" in the proof tree associated to a given higher-level subgoal. After its proof is completed, `lem-try2` is used to close the proof of the subgoal `lem-crit2`, which, in turn, is used (twice) to close the proof of the subgoal `mutex2@0`.

Of course, the subgoal `mutex2@0` was only the second subgoal of the main goal `mutex` depicted in Figure 5 (Page 22). There are seven other such subgoals. Fortunately, four of them are inductive, hence, automatic induction, rewriting, and decision procedures prove them. The three remaining ones do need auxiliary subgoals, but the already proved `lem-try2`, `lem-crit2`, or symmetrical statements obtained from them by replacing the second process by the first one are enough to prove them as well. Here, the user's inspiration in choosing the adequate auxiliary subgoals was important in obtaining a (relatively) simple proof. The complete Maude specification and ITP proof script for this example are available at `http://www.irisa.fr/vertecs/Equipe/Rusu/itp/mutex2`.

5.2 Executability issues

We now discuss some executability issues and show that we can deal with some specifications that are not executable by Maude. Here, by *executability* we mean restrictions imposed in Maude to MEL and RL theories that allow for effective rewriting. These restrictions concern occurrences of free variables[6]:

1. there are no supplementary variables in the conditions of memberships with respect to membership itself; i.e., if $(\forall X)t : s \ \textit{if} \ C$ is a conditional membership, then the free variables in the condition C are among those in the term t: we write as usual $\mathit{vars}(C) \subseteq \mathit{vars}(t)$;

[6] We do not discuss the *confluence/coherence* restrictions [2], which do not concern us here.

2. For equations $(\forall X) l = r \ \textit{if}\ C$, the restriction is $\mathit{vars}(r) \cup \mathit{vars}(C) \subseteq \mathit{vars}(l)$, i.e., there are no supplementary variables in the conditions and right-hand-sides of equations with respect to the left-hand sides[7];

3. A similar restriction holds for rewrite rules $(\forall X) l \rightarrow r \ \textit{if}\ C$, i.e., $\mathit{vars}(r) \cup \mathit{vars}(C) \subseteq \mathit{vars}(l)$. In this way, a match of the left-hand side induces a match of the condition and of the right-hand side.

For the 2-process Bakery algorithm we have actually already dealt with problems posed by these restrictions. Indeed, if we had specified the transition of, e.g., the first process going from `Critical` to `Sleep` simply as

```
rl [Trans14bis] : < Critical, l2, t1, t2 > =>  < Sleep, l2, 0, t2 > .
```

which is an (unconditional) rewrite rule satisfying the above executability restriction, then, our approach would have generated the membership

```
 < Sleep, l2, 0, t2 > : Reachable if < Critical, l2, t1, t2 > : Reachable .
```

in which `t1` violates the restriction $\mathit{vars}(C) \subseteq \mathit{vars}(t)$! This is why we wrote

```
eq Trans14(< Critical, l2, t1, t2 > ) = < Sleep, l2, 0, t2 >  .
rl [Trans14] : < Critical, l2, t1, t2 > => Trans14(< Critical, l2, t1, t2 >) .
```

which generated the following membership (cf. Figure 4, Page 19)

```
cmb Trans14(< Critical, l2, t1, t2 >) if < Critical, l2, t1, t2 > .
```

which satisfies the constraint; actually, it satisfies $\mathit{vars}(C) = \mathit{vars}(t)$.

This solution, which consists in adding new functions $\texttt{trans}_{i,j}$ as shown above, solves the issues posed by the executability restrictions for the conditional memberships generated by our approach from the rewrite rules. But a closer look at the problem shows that the same idea can be used for generating executable conditional memberships, even for rules $(\forall X) l \Rightarrow r \ \textit{if}\ C$ that are *not* executable! Indeed, such rules have the form

$$(\forall X, Y, Z, U)\, l(X, Y) \Rightarrow r(X, Z) \ \textit{if}\ C(X, U) \tag{8}$$

where the (possibly empty) set X denotes the free variables that eveywhere in the rule. X is disjoint from Y, Z, U; these three sets denote the free variables occurring in the left-hand side, the right-hand side, and the condition, respectively. We translate the rule (8) by the following equation and membership, which do satisfy the respective executability restrictions:

$$(\forall X, Y, Z, U)\, \mathit{trans}(X, Z, U, l(X, Y)) = r(X, Z)$$
$$(\forall X, Y, Z, U)\, \mathit{trans}(X, Z, U, l(X, Y)) : \mathit{Reachable}\ \textit{if}\ l(X, Y) : \mathit{Reachable} \wedge C(X, U)$$

What does our ability to encode non-executable rules of the form (8) give us in practice? There are at least two interesting applications:

[7] There are some ways around these restrictions in Maude [2] but ITP does not accept them.

- *parametric systems*, which consist of a parametric number n of identical processes. Such systems are described by rules of the form $(\forall X, \forall Y, \forall i)\, l(X,Y) \Rightarrow r(X,i)$ *if* $C(X,i)$, i.e., $Z = U = \{i\}$ in (8), where i is a natural-number index. We study a parametric system in Section 5.3.

- *open systems*: these are systems that interact with their environment *via* some some shared variables or communication channels. The set of variables $Z \cup U$ in (8) may models either inputs from the environment (that satisfy the assumption C) or outputs to the environment (then, the condition C can be seen as a guarantee). Open systems and assume-guarantee verification are an interesting idea for future work.

5.3 Verifying an n-process Bakery Algorithm

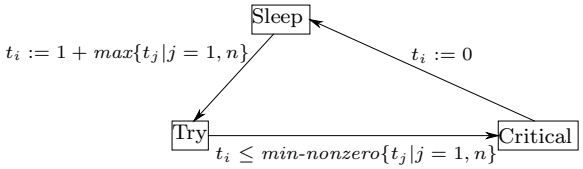


Figure 9: The i-th process in the n-process Bakery algorithm

The n-process Bakery algorithm considered here is a straightforward generalisation of the 2-process version. Each process alternates between *Sleep*, *Try*, and *Critical* locations, and has a ticket denoted by t_i for the i-th process (cf. Figure 9). When going from *Sleep* to *Try*, process number i sets its ticket t_i to the maximum over all tickets plus one. The condition to enter the *Critical* locations is that the current ticket is less or equal to the *minimum non-zero* of the multiset of tickets, where the minimum non-zero of a multiset S of natural numbers is defined by *min-nonzero(S)* $= 0$ if $\forall x.x \in S \Rightarrow x = 0$ and *min-nonzero*$(S) = min(S \setminus \{0\}))$ otherwise. Here, *min* denotes the smallest element of a (nonempty) multiset, and $S \setminus \{0\}$ denotes the multiset obtain by removing all copies of 0 from S. This means that the process entering *Critical* has the smallest ticket among all processes that are trying to enter (i.e., those with nonzero tickets). Finally, each process resets its ticket to zero when going back to *Sleep*.

The idea of the proof of mutual exclusion to *Critical* is quite simple. We prove two auxiliary lemmas:

- the tickets of processes in locations *Try* or *Critical* are all *non-zero*;

- the tickets of processes in locations *Try* or *Critical* are all *distinct*.

The first auxiliary lemma holds, because, intuitively, entering *Try* sets a ticket to a non-zero value, and entering *Critical* does not modify the ticket. The second lemma holds because of

the way the tickets are assigned from the transition from *Sleep* to *Critical*, to the *maximum* value over all tickets plus one. And finally, mutual exclusion holds because, by the first lemma, the minimum nonzero over the set of (non-zero) tickets of all processes in locations *Try* or *Critical*, coincides with the (usual) minimum; and the minimum of a multi-set in which all elements are distinct is unique; hence, an unique process, holding that unique smallest ticket, may enter the critical section at any time.

Modelling and verifying the protocol in Maude/ITP. We first define a *local state* as a pair consisting of a location and a natural number (the ticket). Accessors to the location and ticket are also defined.

```
--- local state : l is the location, n is the ticket
op <_,_> : Loc? Int? -> LocalState? [ctor] .
cmb < l, n > : LocalState if l : Loc /\ n : Nat .
op getLoc : LocalState? -> Loc? .
eq getLoc(< l, n > ) = l .
op getTic : LocalState? -> Int? .
eq getTic(< l, n > ) = n .
```

Like for the 2-process version, `Loc?` and `Loc` denote, respectively, the *kind* and the *sort* of locations; and `Nat?`, resp. `Nat` denote the kind and respectively the sort of natural numbers. The "`?`" notation applies to all the kinds used in the example. Then, a *state* can be: either a local state (denoted using a subsorting relation, `LocalState < State`); or the composition, denoted by the infix operation "`;`", of a local state and another state:

```
--- global state = nonempty list of local states
op _;_ : LocalState? State? -> State? [ctor] .
cmb ls ; st' : State if ls : LocalState /\ st' : State .
```

Next, we define the *size* of a state as its number of processes, denoted by `dim()`, and accessors and modifiers for states, i,.e., `readtAt(st,i)` returns the i-th local state of the state `st`, and `writeAt(st,i,ls)` returns the state `st` whose i-th local state has been replaced by `st`. The maximum `max(st)` and minimum non-zero `min-nonzero(st)` over the tickets of all the local states in a state `st` are also defined. Finally, the four transitions of the system, parameretised by the component that performs them, are defined. Here is, for example, the definition of the transition of process number `i` that goes from `Sleep` to `Try`:

```
op trans1 : Int? State? -> State? .
eq trans1(i,st) = writeAt(st, i, < Try , 1 +  max(st) > ) .
--- crl {st} => {trans1(i,st)} if getLoc(readAt(st,i)) = Sleep
--- /\ i : Nat /\ i < dim(st) /\ getLoc(readAt(st,i)) = Sleep .
```

The rewrite rule that actually fires the transition has been commented out, because it is not executable (cf. Section 5.2). However, the conditional membership that our approach generates from that rule is executable:

```
cmb trans1(i,st) : Reachable if st : Reachable /\  i : Nat /\
i < dim(st) /\ getLoc(readAt(st,i)) = Sleep .
```

The three other rules are written in a similar manner, and generate the corresponding conditional memberships, which, together with the following definition of the (infinitely many) initial states using a non-ground term:

```
op Init : Int? -> State? .
--- initial states = n processes are sleeping
eq Init(0) = < Sleep, 0 > .
eq Init(s n) = < Sleep, 0 >  ; Init(n) .
--- initial state is reachable (for n processes)
 cmb Init(n) : Reachable if n : Nat .
```

provides an inductive definition of all states reachable from the term `Init`.

The *mutual exclusion property* is expressed as the following ITP goal:

```
(goal mutex : BAKERY-N |-
  (A{st:State? ; i:Int? ; j:Int?}
     ((((st:State?) : Reachable) &  ((i:Int?) : Nat) &
         ((i:Int? < dim(st:State?)) = (true)) &((j:Int?) : Nat) &
         ((j:Int? < dim(st:State?)) = (true)) &
         ((getLoc(readAt(st:State?, i:Int?))) = (Critical)) &
         ((getLoc(readAt(st:State?, j:Int?))) = (Critical)))
      => ((i:Int?) = (j:Int?)))) .)
```

The `mutex` goal says that in all reachable states, a certain *state predicate* (in the sense of Definition 14) holds ; the state predicate characterises all states `st` in mutual exclusion, i.e., if two local states at positions `i < dim(st)` and `j < dim(st)` are both in the `Critical` location then necessarily `i = j`.

The ITP proof of the mutual exclusion property goes much like the informal proof sketched at the beginning of this section. The user states and proves two auxiliary lemmas. The first one, `nonZero-tickets`, says that in all locations except `Sleep`, tickets are strictly positive. The predicate `eqNat` is just equality over natural numbers; we use it to disable ITP's rewriting of equals by equals when it is not necessary.

```
(goal nonZero-tickets :  BAKERY-N |-
 (A{st:State? ; i:Int?}
   ((((st:State?): Reachable) & ((i:Int?) : Nat) &
     ((i:Int? < dim(st:State?)) = (true)) &
     ((isSleep(getLoc(readAt(st:State?, i:Int?)))) = (false)))
   =>  ((getTic(readAt(st:State?, i:Int?)) > 0) = (true)))) .)
```

The second auxiliary lemma, `allDistinct-tickets` (below), says that the processes in all locations except `Sleep` cannot have equal tickets:

```
(goal allDistinct-tickets : BAKERY-N |-
 (A{st:State? ; i:Int? ; j:Int?}
   ((((st:State?): Reachable) & ((i:Int?) : Nat) &
     ((i:Int? < dim(st:State?)) = (true)) &  ((j:Int?) : Nat) &
     ((j:Int? < dim(st:State?)) = (true)) &
     ((isSleep(getLoc(readAt(st:State?, i:Int?)))) = (false)) &
     ((isSleep(getLoc(readAt(st:State?, j:Int?)))) = (false)) &
     ((i:Int? < j:Int?) = (true)))
   => ((eqNat(getTic(readAt(st:State?, i:Int?)),
              getTic(readAt(st:State?, j:Int?)))) = (false)))) .)
```

The ITP proofs for these goals all follow the same pattern, already given at the beginning of this section. First, induction over the `Reachable` sort generates five subgoals: one for the initial state, and four for the four transitions. The base case (for the initial state) is automatically solved by `(auto)`. The inductive steps are solved by combinations of `(auto)` and case analysis. However, unlike the simple 2-process version, we had to prove quite a few theorems about *all* states (not only the *reachable* ones - only the three theorems shown above are specific to reachable states). This is because ITP needs to know all there is to know about the `max()`, `min-nonzero()`, `readAt()`, and `writeAt()` operators that we have defined on states. About forty such theorems were stated and proved. The complete ITP sources for the example are available at `http://www.irisa.fr/vertecs/Equipe/Rusu/itp/mutex-n`.

6 Conclusion

Automatic state-space exploration, model checking for finite instances, abstraction for reducing infinite-state systems to finite ones, and interactive theorem proving for infinite-state systems are well-known verification techniques. Except for theorem proving for rewriting-logic specifications, all these techniques exist in the Maude environment, and our contribution adds the missing part.

The approach is based on an automatic translation of Rewriting Logic into Membership Equational Logic, and on using the ITP tool on the resulting Membership Equational Logic theory. Our approach is able to deal with systems that are not directly executable in Maude due to supplementary variables in the right-hand sides of rules.

The proposed approach also has some limitations. Like all approaches based on theorem proving, it requires user input and expertise with the system under verification. In our experience, the user gains such expertise during the verification process and benefits from feedback that the prover provides when it fails to prove a given subgoal due to insufficient information. In such cases the user typically "sees" from the pending subgoal and from its proof context what the missing information is, and is able to provide it under the form of an auxiliary subgoal or lemma that, when proved, allows to settle the pending subgoal and to progress in the proof.

There are some limitations that are specific to the current approach as well. The main limitation arises from the constraint that we have imposed to state predicates, that they

should be formulas of membership equational logic (that Maude can also understand). This may sometimes lead to awkward formulations, for instance, the inability to deal with existential quantifiers imposes to replace existential quantification over a variable by an actual construction of a witness for that variable, which requires to define new operations and to prove properties about them. It is an interesting question whether this construction can always be done. We note that ITP allows for more general properties, expressed in Many-Kinded First-order Logic with Equality, which does allow for existential quantifiers; but using the ITP commands for, e.g., instantiating existential quantifiers currently requires considerable ITP expertise.

The approach is illustrated on a 2-process Bakery algorithm, and then on an n-process generalisation of the algorithm. The results are encouraging, and we believe that using our approach based on the prototype ITP prover is competitive with the use of more mature provers such as the PVS theorem prover [15]. One distinctive feature is of our approach is that it is seamleasly integrated with Maude, and thus it offers access to all of Maude's set of tools, including state-space exploration, model checking, and equational abstractions. Hence, the proposed approach is a step towards integrated formal verification as advocated by Rushby [16], but within the Maude framework.

In the future we are planning to experiment on a significant case study [12] a combination of our theorem-proving approach with model checking and equational abstractions [8] to automatically prove as many auxiliary invariants as possible during the verification effort.

References

[1] N. Martí-Oliet and J. Meseguer. Rewriting logic: roadmap and bibliography. *Theor. Comput. Sci.*, 285(2):121–154, 2002.

[2] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C.L̃. Talcott. *All About Maude, A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[3] P. Borovansky, C. Kirchner, H. Kirchner, P.Ẽ. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. In *Proc. of the First Int. Workshop on Rewriting Logic*, volume 4. Elsevier, 1996.

[4] R. Diaconescu and K. Futatsugi. Logical foundations of cafeobj. *Theoretical Computer Science*, 285(2):289–318, 2002.

[5] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *WADT*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1997.

[6] R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. *Theor. Comput. Sci.*, 360(1-3):386–414, 2006.

[7] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker. *Electr. Notes Theor. Comput. Sci.*, 71, 2002.

[8] J. Meseguer, M. Palomino, and N. Martí-Oliet. Equational abstractions. In F. Baader, editor, *CADE*, volume 2741 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2003.

[9] M. Clavel, M. Palomino, and A. Riesco. Introducing the ITP tool: a tutorial. *J. Universal Computer Science*, 12(11):1618–1650, 2006.

[10] J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to the verification of cryptographic protocols. In N. Martí-Oliet, editor, *International Workshop on Rewriting Logic and Applications*, volume 117 of *Electronic Notes in Theoretical Computer Science*, Amsterdam, 2004. Elsevier.

[11] G. Roşu. *Hidden Logic*. PhD thesis, Univ. California at San Diego, 2000.

[12] W. Kong, K. Ogata, and K. Futatsugi. Algebraic approaches to the formal analysis of an electronic purse system. In J. Davies and J. Gibbons, editors, *Integrated Formal Methods, 6th International Conference IFM 2007, Oxford, UK*, volume 4591 of *Lecture Notes in Computer Science*, pages 393–412. Springer, 2007.

[13] M. Clavel, M. Palomino, and J. Santa-Cruz. Integrating decision procedures in reflective rewriting-based theorem provers. In *4th International Workshop on Reduction Strategies in Rewriting and Programming*, pages 15–24, 2004.

[14] E.M̃. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[15] S. Owre, J. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[16] J. Rushby. Integrated formal verification: Using model checking with automated abstraction, invariant generation, and theorem proving. In D. Dams, R. Gerth, S. Leue, and M. Massink, editors, *Theoretical and Practical Aspects of SPIN Model Checking: 5th and 6th International SPIN Workshops*, volume 1680, pages 1–11, Trento, Italy, and Toulouse, France, 1999. Springer-Verlag.

Appendix: intermediary results and proofs

Proofs of results from Section 2.2

The main results to be proved are Propositions 1 and 2. They are reproduced below. Some intermediary lemmas are necessary.

Consider an equation of the form $(\forall X)t = t'$ with $t, t' \in T_\Sigma(X)$. To check the validity of such an equation in the initial model (cf. the *Semantics* paragraph), we need *kind-preserving valuations of the variables*, i.e., kind-preserving functions from the variables X to the set of equivalence classes $T_{\Sigma/E}$ of ground terms. Using a standard (exponentiation) notation for functions from X to T_Σ, this set of functions is denoted $T^X_{\Sigma/E}$.

The first lemma shows that there is a surjection between the set T^X_Σ of *ground substitutions* of variables X, i.e., (kind-preserving) functions from X to T_Σ, and the set $T^X_{\Sigma/E}$:

Lemma 1 *The function* $[\_]_E : T^X_\Sigma \mapsto T^X_{\Sigma/E}$, *defined, for all* $\sigma \in T^X_\Sigma$, *by* $([\sigma]_E)(x) = [x\sigma]_E$ *for all* $x \in X$, *is well defined and is a surjection.* □

Proof. For the well definedness we note that $[\_]_E$ does send substitutions in T^X_Σ to valuations in $T^X_{\Sigma/E}$. For proving the surjectiveness of the function $[\_]_E$, let $\eta \in T^X_{\Sigma/E}$ be an arbitrary valuation: $\eta(x) = [t_x]_E$ for all $x \in X$, and consider the ground substitution given by $x\sigma = t_x$ for all $x \in X$; then, clearly, $[\sigma]_E = \eta$, as for all $x \in X$, $([\sigma]_E)(x) = [x\sigma]_E = [t_x]_E = \eta(x)$. □

The next lemma is an "upgrading" of Lemma 1 from variables to terms:

Lemma 2 *For all* $t \in T_\Sigma(X)$ *and* $\sigma \in T^X_\Sigma$, $[\sigma]_E([t]_E) = [t\sigma]_E$. □

Proof. By induction on the structure of term t. If t is a variable then the conclusion follows by Lemma 1. Otherwise, $t = f(t_1, \ldots, t_n)$ for some n-ary function symbol $f \in \Sigma$ (with $n \geq 0$). Then,

$$[\sigma]_E([t]_E) = [\sigma]_E([f(t_1, \ldots, t_n)]_E)$$

By the homomorphic nature of the valuation $[\sigma]_E$ in the inital model:

$$[\sigma]_E([f(t_1, \ldots, t_n)]_E) = A_f([\sigma]_E(t_1), \ldots [\sigma]_E(t_1))$$

By induction hypothesis:

$$A_f([\sigma]_E(t_1), \ldots [\sigma]_E(t_1)) = A_f([t_1\sigma]_E, \ldots, [t_n\sigma]_E)$$

By the definition of A_f in the initial model:

$$A_f([t_1\sigma]_E, \ldots, [t_n\sigma]_E) = [f(t_1\sigma, \ldots, t_n\sigma)]_E$$

By the definition of substitutions:

$$[f(t_1\sigma, \ldots, t_n\sigma)]_E = [(f(t_1, \ldots, t_n)\sigma)]_E = [t\sigma]_E.$$

and the conclusion follows by transitivity of equality. □

The next lemma characterises validity in the initial model for atomic formulas (equations or memberships): an atomic predicate is valid in the initial model iff all its ground instances are valid in all models.

Lemma 3 *Let φ denote either an equation $t = t'$ or a membership $t : s$, with $t, t' \in T_\Sigma(X)$ and $s \in S_k$ for some kind k. For a ground substitution $\sigma : X \mapsto T_\Sigma$, we let $\varphi\sigma$ denote, respectively, the equality $t\sigma = t'\sigma$, or the membership $t\sigma : s$. Then, $E \vdash_{ind} (\forall X)\varphi$ iff $E \vdash (\forall \emptyset)\varphi\sigma$ for all $\sigma \in T_\Sigma^X$.* □

Proof.

- We first consider the case when φ is an equality $t = t'$.

  (⇒) We have, by Definition 2 of the initial model and by the notion of semantic validity, $E \vdash_{ind} (\forall X)t = t'$ iff for all valuations $\eta \in T^X_{\Sigma/E}$, $\eta([t]_E) = \eta([t']_E)$. Consider then an arbitrary ground substitution $\sigma \in T_\Sigma^X$. Then, by Lemma 1, $\eta = [\sigma]_E$ is a valuation in $T^X_{\Sigma/E}$. Hence, $[\sigma]_E([t]_E) = [\sigma]_E([t']_E)$. But by Lemma 2, $[\sigma]_E([t]_E) = [t\sigma]_E$ and $[\sigma]_E([t']_E) = [t'\sigma]_E$. By transitivity if equality, we obtain $[t\sigma]_E = [t'\sigma]_E$, i.e., $E \vdash (\forall\emptyset)t\sigma = t'\sigma$ for any (arbitrary) ground substitution $\sigma \in T_\Sigma^X$, which proves (⇒).

  (⇐) Assume that $E \vdash (\forall)t\sigma = t'\sigma$ for all ground substitutions $\sigma \in T_\Sigma^X$, and consider an arbitrary valuation $\eta \in T^X_{\Sigma/E}$. Then, by Lemma 1, we can find a ground substitution $\sigma \in T_\Sigma^X$ such that $\eta = [\sigma]_E$. Now, $E \vdash (\forall\emptyset)t\sigma = t'\sigma$ implies $[t\sigma]_E = [t'\sigma]_E$, which, by Lemma 2 implies $[\sigma]_E([t]_E) = [\sigma]_E([t']_E)$, i.e., $\eta([t]_E) = \eta([t']_E)$ for any (arbitrary) valuation $\eta \in T^X_{\Sigma/E}$; by Definition 2 of the initial model and by the notion of semantic validity we then have $E \vdash_{ind} (\forall X)t = t'$, and the proof is done when φ is an equation.

- Next, we consider the case where φ is a membership $t : s$. Let $s \in S_k$ for some kind k. Remember that, by Definition 2 of the initial model and by the notion of semantic validity, $E \vdash_{ind} (\forall X)t : s$ iff, for all valuations $\eta \in T^X_{\Sigma/E}$: $\eta([t]_E) \in \{[t']_E \in T_{\Sigma,k/E} | E \vdash (\forall\emptyset)t' : s\}$.

  (⇒) Let then σ be an arbitrary ground substitution in $\sigma \in T_\Sigma^X$. Then, by Lemma 1 $[\sigma]_E \in T^X_{\Sigma/E}$, and then, by the above observation, $[\sigma]_E([t]_E) \in \{[t']_E \in T_{\Sigma,k/E} | E \vdash (\forall\emptyset)t' : s\}$. But by Lemma 2, $[\sigma]_E([t]_E) = [t\sigma]_E$, and then $[t\sigma]_E \in \{[t']_E \in T_{\Sigma,k/E} | E \vdash (\forall\emptyset)t' : s\}$, which just means that $E \vdash (\forall\emptyset)t\sigma : s$ for any (arbitrary) ground substitution $\sigma \in T_\Sigma^X$, which proves (⇒).

  (⇐) Conversely, assume that $E \vdash (\forall\emptyset)t\sigma : s$ for all ground substitutions $\sigma \in T_\Sigma^X$, and consider an arbitrary valuation $\eta \in T^X_{\Sigma/E}$. Then, by Lemma 1, we can find a ground substitution $\sigma \in T_\Sigma^X$ such that $\eta = [\sigma]_E$, and $\eta([t]_E) = [t\sigma]_E \in \{[t']_E \in T_{\Sigma,k/E} | E \vdash (\forall\emptyset)t'\sigma : s\}$, which implies $E \vdash_{ind} (\forall X)t : s$ and concludes the proof. □

We are now ready to prove the first Proposition in Section 2.2:

Proposition 1 *Consider a* MEL *sentence* $(\forall X)\ \psi = (\forall X)\ \psi_0$ *if* $\psi_1 \wedge \ldots \wedge \psi_n$, *where all the* ψ_i, *for* $i \in [0, n]$, *are atomic formulas of a* MEL *theory* (Σ, E). *Consider also a ground atomic membership* $(\forall \emptyset) t : s$ *with* $t \in T_\Sigma(X)$. *Then, the implication* $(\forall \emptyset) t : s \Rightarrow (\forall X)\psi$ *is a* MEL *sentence of* (Σ, E); *and* $E \vdash_{ind} ((\forall \emptyset) t : s \Rightarrow (\forall X)\psi)$ *iff the following logical implication holds:* $[E \vdash (\forall \emptyset) t : s \Rightarrow E \vdash_{ind} (\forall X)\psi]$. □

Proof. The first part of the result amounts to observing that $(\forall \emptyset) t : s \Rightarrow (\forall X)\psi$ is a synonym for the MEL sentence $(\forall X)\ \psi_0$ *if* $t : s \wedge \psi_1 \wedge \ldots \wedge \psi_n$, perhaps after renaming some of the variables in X to avoid undesired bindings in t. For the second part, we introduce the following notations. We denote the implication ψ_0 *if* $t : s \wedge \psi_1 \wedge \ldots \wedge \psi_n$ by ψ'. For $\eta \in T^X_{\Sigma/E}$ and an atomic formula $(\forall X)\varphi_i$, $\eta(\varphi_i)$ denotes $\eta([t]_E) = \eta([t']_E)$ if φ_i is an equation $t = t'$, or, if η is a membership $t : s$, $\eta(\varphi)$ denotes $\eta([t]_E) \in A_s$.

Then, we know by Definition 2 and by the *Semantics* paragraph that $E \vdash_{ind} (\forall X)\psi'$ iff

$$\forall \eta \in T^X_{\Sigma/E}.\ \left(\left[[t]_E \in A_s \wedge \bigwedge_{i=1}^{n} \eta(\psi_i)\right] \Rightarrow \eta(\psi_0\right)$$

which is logically equivalent to

$$\forall \eta \in T^X_{\Sigma/E}.\ \left([t]_E \in A_s \Rightarrow \left[\left(\bigwedge_{i=1}^{n} \eta(\psi_i)\right) \Rightarrow \eta(\psi_0)\right]\right)$$

which is, since η is not involved in the left-hand side of the first implication, is equivalent to

$$[t]_E \in A_s \Rightarrow \left(\forall \eta \in T^X_{\Sigma/E}.\ \left[\left(\bigwedge_{i=1}^{n} \eta(\psi_i)\right) \Rightarrow \eta(\psi_0)\right]\right)$$

which, by Definition 2 of validity in initial models, is just

$$[t]_E \in A_s \Rightarrow (E \vdash_{ind} (\forall X)\psi)$$

which, by Definition 2 again, is just

$$(E \vdash_{ind} (\forall \emptyset) t : s) \Rightarrow (E \vdash_{ind} (\forall X)\psi)$$

and the result follows from the fact that $E \vdash_{ind} (\forall \emptyset) t : s$ iff $E \vdash (\forall \emptyset) t : s$ for ground terms t. □

The remaining result to prove from Section 2.2 is Proposition 2. Again, some intermediary results are needed. The next lemma deals with the validity of *ground* MEL sentences in the initial model:

Lemma 4 *Consider a ground* MEL *sentence* $(\forall \emptyset)\ \psi = (\forall \emptyset)\ \psi_0$ *if* $\psi_1 \wedge \ldots \wedge \psi_n$, *where all the* ψ_i *are ground* MEL *atomic formulas, for* $i \in [0, n]$. *Then,* $E \vdash_{ind} (\forall \emptyset)\psi$ *iff the following implication holds:* $[\bigwedge_{i=1}^{n} E \vdash (\forall \emptyset)\psi_i] \Rightarrow E \vdash (\forall \emptyset)\psi_0$. □

Proof. Directly from the definitions of validity and of the initial model. □

The following lemma is a generalisation of Lemma 4 to MEL sentences.

Lemma 5 *Let $(\forall X)\varphi = (\forall X)\ \varphi_0$ if $\varphi_1 \wedge \ldots \wedge \varphi_n$, be a* MEL *sentence of a* MEL *theory (Σ, E). For a ground substitution $\sigma : X \mapsto T_\Sigma$, we let $\varphi\sigma$ denote the ground* MEL *sentence $(\forall\emptyset)\ \varphi_0\sigma$ if $(\bigwedge_{i=1}^n \varphi_i\sigma)$. Then, $E \vdash_{ind} (\forall X)\varphi$ iff $E \vdash_{ind} (\forall\emptyset)\varphi\sigma$ for all $\sigma \in T_\Sigma^X$.* □

Proof (sketch). We use the notations from the proof of Proposition 1: for $\eta \in T^X_{\Sigma/E}$ and an atomic formula $(\forall X)\varphi_i$, $\eta(\varphi_i)$ denotes $\eta([t]_E) = \eta([t']_E)$ if φ_i is an equation $t = t'$, or, if η is a membership $t : s$, $\eta(\varphi)$ denotes $\eta([t]_E) \in A_s$. Using Lemmas 1 and 3 and the definition of validity in the initial model, is not hard to check that

(†) for all ground substitutions $\sigma \in T_\Sigma^X$, $[\sigma]_E(\varphi_i)$ iff $E \vdash (\forall\emptyset)\varphi_i\sigma$.

Then, $E \vdash_{ind} (\forall X)\varphi$ iff (by the definition of validity in the initial model)

$$\forall \eta \in T^X_{\Sigma/E}.\ \left(\left[\bigwedge_{i=1}^n \eta(\varphi_i)\right] \Rightarrow \eta(\varphi_0)\right)$$

iff (by the surjection established in Lemma 1)

$$\forall \sigma \in T_\Sigma^X.\ \left(\left[\bigwedge_{i=1}^n [\sigma]_E(\varphi_i)\right] \Rightarrow [\sigma]_E(\varphi_0)\right)$$

iff (by the observation (†) above)

$$\forall \sigma \in T_\Sigma^X.\ \left(\left[\bigwedge_{i=1}^n E \vdash (\forall\emptyset)\varphi_i\sigma\right] \Rightarrow E \vdash (\forall\emptyset)\varphi_0\sigma\right)$$

iff (by Lemma 4) $\forall \sigma \in T_\Sigma^X.\, E \vdash_{ind} (\forall\emptyset)\varphi\sigma$, and the proof is done. □

Proposition 2 *Let $(\forall x : [k])(\forall X)\varphi$ be a sentence of a* MEL *theory (Σ, E), for some kind k of the theory, such that $x \notin X$. Then, $E \vdash_{ind} (\forall x : [k])(\forall X)\varphi$ iff $E \vdash_{ind} (\forall X)\varphi[t/x]$ for all ground terms $t \in T_{\Sigma,k}$.* □

Proof. By Lemma 5, $E \vdash_{ind} (\forall x : [k])(\forall X)\varphi$ iff for all ground valuation $\sigma \in T_\Sigma^{\{x\}\cup X}$, $E \vdash (\forall\emptyset)\varphi\sigma$. Now, the valuation $\sigma \in T_\Sigma^{\{x\}\cup X}$ can be decomposed into

- a ground valuation σ^x, which associates to the variable x some ground term $\sigma^x(x) \in T_{\Sigma,k}$ and
- a ground valuation σ^X for the rest of the variables X.

Then, $E \vdash_{ind} (\forall x : [k])(\forall X)\varphi$ iff $\forall \sigma \in T_\Sigma^{\{x\}\cup X}.\, E \vdash (\forall\emptyset)\varphi\sigma$ iff

$$\forall \sigma^x(x) \in T_{\Sigma,k}.\, \forall \sigma^X \in T_\Sigma^X.\, E \vdash (\forall\emptyset)(\varphi\sigma^x)\sigma^X$$

By renaming the *term* $\sigma^x(x) \in T_{\Sigma,k}$ into t,

$$\forall t \in T_{\Sigma,k}. [\forall \sigma^X \in T_\Sigma^X. E \vdash (\forall \emptyset)(\varphi[t/x])\sigma^X]$$

and the conclusion follows from the fact that the sub-formula (above): $[\forall \sigma^X \in T_\Sigma^X. E \vdash (\forall \emptyset)(\varphi[t/x])\sigma^X]$ is equivalent to $E \vdash_{ind} (\forall X)\varphi[t/x]$. □

Proofs of results from Section 2.3

The main result here is Proposition 3. Several intermediary lemmas are needed; they mainly deal with deduction using membership assertions.

Lemma 6 *Consider a* MEL *theory* $\mathcal{M} = (\Sigma, E \cup \{(\forall X)t : s\})$ *and a sort* s *of* $\mathcal{M}$ *such that all the defining assertions of* s *in* $\mathcal{D}(s) \cap E$ *are inductive. Assume that there exists a a derivation of the entailment* $E \cup \{(\forall X)t_1 : s\} \vdash (\forall X)t_2 : s$*, obtained by applying any of the deduction rules given in Definitions 1 and 3, with the restriction that the Replacement*$_2$ *rule cannot be used with memberships in* $\mathcal{D}(s)$*. Then* $E \cup \{(\forall X)t_1 : s\}) \vdash (\forall X)t_1 = t_2$. □

Proof. By complete induction on the length $n \geq 1$ of the derivation of $E' \vdash (\forall X)t_2 : s$, where $E' = E \cup \{(\forall X)t_1 : s\}$. The last step in the derivation is either an application of the *Membership* or of the *Membership'* rules, because these are the only allowed rules whose conclusion can be a membership in the sort s (as we have forbidden by hypothesis of our lemma the use of the *Replacement*$_2$ rule with memberships in $\mathcal{D}(s)$).

- if the last step in the derivation is an application of the *Membership* rule, then there exists a term $t \in T_\Sigma(X)$ such that $E' \vdash (\forall X)t : s$ and $E' \vdash (\forall X)t_2 = t$, and each of these two entailments have a derivation of length $< n$. We use the induction hypothesis for $E' \vdash (\forall X)t : s$ and obtain $E' \vdash (\forall X)t_1 = t$, and, using the *Transitivity* and *Symmetry* rules, we obtain $E' \vdash (\forall X)t_1 = t_2$, and the proof is done in this case.
- if the last step in the derivation is an application of the *Membership'* rule, we obtain $(\forall X)t_2 : s \in E \cup \{(\forall X)t_1 : s\}$, which implies that $(\forall X)t_2 : s$ actually *is* $(\forall X)t_1 : s$ since, by hypothesis, all the other defining memberships for s in E are inductive, but $(\forall X)t_2 : s$ is not. Hence, the terms t_1 and t_2 are syntactically equal[8], which implies by *Reflexivity* that $E' \vdash (\forall X)t_1 = t_2$, and the proof is done. □

The next lemma states that the defining assertions of a simply inductive sort s do not participate in the inference of equalities, or of memberships in sorts other than s. This result will be used in several other proofs.

Lemma 7 *Consider a* MEL *theory* $\mathcal{M} = (\Sigma, E)$ *and a simply inductive sort* s *of* $\mathcal{M}$*. Assume that there exists a a derivation of the entailment* $E \cup \{(\forall X)t : s \text{ if } C\} \vdash e$*, where* e *is either a membership* $(\forall Y)t' : s'$ *with* $s' \neq s$*, or an equation* $(\forall Y)t_1 = t_2$*. Then,* $E \vdash e$. □

[8] We denote syntactical equality between tow terms t_1 and t_2 by $t_1 = t_2$, unlike equality in a MEL theory (Σ, E), which is denoted by $E \vdash (\forall X)t_1 = t_2$.

Proof. We let $E' = E \cup \{(\forall X)t : s \text{ if } C\}$. The proof goes by complete induction on the length $n \geq 1$ of the derivation of $E' \vdash e$.

- Let us first consider the case where e is a membership $(\forall Y)t' : s'$. Consider a derivation in $n \geq 1$ steps of $E' \vdash (\forall Y)t' : s'$. The derivation terminates either with the *Membership*, *Membership'*, or the *Replacement*$_2$ rules. We do not need to consider the *Membership'* because, here, the *Replacement*$_2$ rule subsumes it.

  - if the considered derivation ends with the *Membership* rule, then there exists a term t'' such that $E' \vdash (\forall Y)t'' : s'$ and $E' \vdash (\forall Y)t'' = t'$, and these entailments are derived in strictly fewer than n steps. The induction hypothesis then gives us $E \vdash (\forall Y)t'' : s'$ and $E \vdash (\forall Y)t'' = t'$, and then we obtain using the *Transitivity* and *Symmetry* rules that $E \vdash (\forall Y)t' : s'$.
  - if the derivation ends with the *Replacement*$_2$ rule, then, there exist a defining membership $(\mu) : (\forall Z)t'' : s' \text{ if } C' \in \mathcal{D}(s')$ and a kind-preserving substitution $\sigma : Z \mapsto T_\Sigma(Y)$ such that $E' \vdash (\forall Y)C'\sigma$ is derived in strictly fewer than n steps, and $t' = t''\sigma$; hence, by induction hypothesis
    - E entails all *memberships* in the condition C'
    - and E entails all *equations* in the condition C'.

    Hence, $E \vdash (\forall Y)C'\sigma$ holds as well, meaning that the membership μ can be applied as well by using just the sentences in E to validate its condition C'. Hence, $E \vdash (\forall Y)t' : s'$ and the proof is done in the case where e is a membership.

- We now briefly consider the case where e is a equality $(\forall Y)t_1 = t_2$. A derivation of $n \geq 1$ steps ends with either *Reflexivity*, *Symmetry*, *Transitivity*, *Congruence*, or *Replacement*$_1$. The first four cases are trivial: in each case, the induction hypothesis tells us that the *premise* of the rule also holds when E is used instead of E'; and then the *conclusion* of each of the rules, used with E instead of E', gives us the expected entailment $E \vdash (\forall Y)t_1 = t_2$. If the last step is *Replacement*$_1$, then, there exist an equation $(e) : (\forall Z)t'_1 = t'_2 \text{ if } C' \in E'$, with $t'_1, t'_2 \in T_\Sigma(Z)$, and a kind-preserving substitution $\sigma : Z \mapsto T_\Sigma(Y)$, such that $E' \vdash (\forall Y)C'\sigma$ can be derived in strictly fewer than n steps, $t_1 = t'_1\sigma$, and $t_2 = t'_2\sigma$; hence, by induction hypothesis
  - E entails all *memberships* in the condition C'
  - and E entails all *equations* in the condition C'.

  Hence, $E \vdash (\forall Y)C'\sigma$ holds as well, meaning that the equation (e) can be applied as well by using just the sentences in E to validate its condition C', which implies $E \vdash (\forall Y)t_1 = t_2$. □

Putting together Lemma 6 and Lemma 7 we obtain Corollary 2 below.

Corollary 2 *Consider a* MEL *theory* $\mathcal{M} = (\Sigma, E \cup \{(\forall X)t : s\})$ *and a simply inductive sort* s *of* $\mathcal{M}$ *such that all the defining assertions of* s *in* $\mathcal{D}(s) \cap E$ *are inductive. Assume that there exists a a derivation of the entailment* $E \cup \{(\forall X)t_1 : s\} \vdash (\forall X)t_2 : s$*, obtained by applying any of the deduction rules given in Definitions 1 and 3, with the restriction that the Replacement*$_2$ *rule cannot be used with memberships in* $\mathcal{D}(s)$*. Then* $E \vdash (\forall X)t_1 = t_2$*.* □

The next lemma says that kind-preserving substitutions preserve sorts:

Lemma 8 *Consider a* MEL *theory* $\mathcal{M} = (\Sigma, E)$*. Then, for all sorts* s *of* $\mathcal{M}$*, all terms* $t \in T_\Sigma(X)$ *and all kind-preserving substitutions* $\sigma : X \mapsto T_\Sigma(Y)$*, if* $E \vdash (\forall X)t : s$*, then* $E \vdash (\forall Y)t\sigma : s$*.*

Proof. By complete induction on the length $n \geq 1$ of the derivation of $E \vdash (\forall X)t : s$. The last step of the derivation is an application of either a *Membership* or of a *Replacement*$_2$ rule.

- if the last step of the derivation of $E \vdash (\forall X)t : s$ is *Membership*, then there exists a term $t' \in T_\Sigma(X)$ such that the entailments $E \vdash (\forall X)t = t'$ and $E \vdash (\forall X)t' : s$ are both derivable in $< n$ steps. We apply the induction hypothesis to the latter entailment and obtain $E \vdash (\forall Y)t'\sigma : s$. On the other hand, from $E \vdash (\forall Y)t = t'$ and the *Replacement*$_1$ rule we obtain $E \vdash (\forall Y)t\sigma = t'\sigma$, and the conclusion follows by the *Membership* rule.
- if the last step of the derivation of $E \vdash (\forall X)t : s$ is *Replacement*$_2$, then there exists a membership (μ) $(\forall Z)t'' : s$ *if* $C \in \mathcal{D}(s)$ and a kind-preserving substitution $\sigma' : Z \mapsto T_\Sigma(X)$ such that $t = t''\sigma'$ and $E \vdash (\forall X)C\sigma'$. We show that $E \vdash (\forall Y)t\sigma : s$ can be obtained by an application of the *Replacement*$_2$ rule with the same membership (μ) but with a different substitution, namely, the composed substitution $\sigma' \circ \sigma : Z \mapsto T_\Sigma(Y)$ (where $\circ$ denotes "reverse" function composition, i.e., σ' is applied first, and σ is applied to the result). For this, remember that the condition C has the form $\bigwedge_{i\in I}(\forall Z)(u_i = v_i) \wedge \bigwedge_{j\in J}(\forall Z)(w_j : s_j)$, for some finite sets of indices I, J, and that $E \vdash (\forall X)C\sigma'$ is a shortcut for $\bigwedge_{i\in I} E \vdash (\forall X)(u_i\sigma' = v_i\sigma') \wedge \bigwedge_{j\in J} E \vdash (\forall X)(w_j\sigma' : s_j)$. Now, $E \vdash (\forall X)C\sigma'$ was obtained in $< n$ steps. By applying the induction hypothesis to each of the entailments $E \vdash (\forall X)(w_j\sigma' : s_j)$, for all $j \in J$, we obtain $E \vdash (\forall Y)(w_j\sigma'\sigma : s_j)$, and by virtue of the *Replacement*$_1$ rule, from each of the entailments $E \vdash (\forall X)(u_i\sigma' = v_i\sigma')$ for all $i \in I$, we obtain $E \vdash (\forall Y)(u_i\sigma'\sigma = v_i\sigma'\sigma)$. Hence, $E \vdash (\forall Y)C\sigma'\sigma$. From $t = t''\sigma'$ we obtain $t\sigma = t''\sigma'\sigma$. Hence, we can apply the *Replacement*$_2$ rule with membership μ and substitution $\sigma' \circ \sigma$ to obtain $E \vdash (\forall Y)t''\sigma'\sigma : s$, i.e., $E \vdash (\forall Y)t\sigma : s$. □

The following lemma is a form of *modus ponens*:

Lemma 9 *Consider a* MEL *theory* $\mathcal{M} = (\Sigma, E)$*, a simply inductive sort* s[9] *of* $\mathcal{M}$*, and terms term* $t, t' \in T_\Sigma(X)$ *such that* $E \vdash (\forall X)t : s$ *and* $E \cup \{(\forall X)t : s\}) \vdash (\forall X)t' : s$*. Then,* $E \vdash (\forall X)t' : s$*.* □

[9] The hypothesis "s is simply inductive" can be dropped, but then we need to strengthen the conclusion of the lemma with entailments of *equations* in order to pass the induction step.

Proof. By complete induction on the length $n \geq 1$ of the derivation of $E \cup \{(\forall X)t : s\}) \vdash (\forall X)t' : s$. The last step of the derivation is an application of either a *Membership* or of a *Replacement*$_2$ rule.

- if the last step of the derivation of $E \cup \{(\forall X)t : s\}) \vdash (\forall X)t' : s$ is *Membership*, then, there exists a term $t'' \in T_\Sigma(X)$ such that both the entailments $E \cup \{(\forall X)t : s\}) \vdash (\forall X)t' = t''$ and $E \cup \{(\forall X)t : s\}) \vdash (\forall X)t'' : s$ can be derived in $< n$ steps. By applying the induction hypothesis to the latter entailment we obtain $E \vdash (\forall X)t'' : s$, and by applying Lemma 7 to $E \cup \{(\forall X)t : s\}) \vdash (\forall X)t' = t''$ we obtain $E \vdash (\forall X)t' = t''$, and the conclusion follows by the *Membership* rule.
- if the last step of the derivation of $E \cup \{(\forall X)t : s\}) \vdash (\forall X)t' : s$ is *Replacement*$_2$, then the rule is applied with a given membership $\mu \in E \cup \{(\forall X)t : s\}$.
  - If $\mu \in E$, then let $(\mu)\ (\forall Z)t'' : s$ *if* C, and there exists also a kind-preserving substitution $\sigma : Z \mapsto T_\Sigma(X)$ such that $t' = t''\sigma$ and $E \cup \{(\forall X)t : s\}) \vdash (\forall X)C\sigma$. The condition C has the form $\bigwedge_{i\in I}(\forall Z)(u_i = v_i) \wedge \bigwedge_{j\in J}(\forall Z)(w_j : s_j)$, for some finite sets of indices I, J, and $E \cup \{(\forall X)t : s\}) \vdash (\forall X)C\sigma$ is a shortcut for $\bigwedge_{i\in I} E \cup \{(\forall X)t : s\}) \vdash (\forall X)(u_i\sigma = v_i\sigma) \wedge \bigwedge_{j\in J} E \cup \{(\forall X)t : s\}) \vdash (\forall X)(w_j\sigma : s_j)$. By applying Lemma 7 to all the entailments $E \cup \{(\forall X)t : s\}) \vdash (\forall X)(u_i\sigma = v_i\sigma)$, we obtain $E \vdash (\forall X)(u_i\sigma = v_i\sigma)$ for all $i \in I$. Next, all the entailments $E \cup \{(\forall X)t : s\}) \vdash (\forall X)(w_j\sigma : s_j)$ were obtained in $< n$ steps, hence, by applying the induction hypothesis we obtain $E \vdash (\forall X)(w_j\sigma : s_j)$ for all $j \in J$. Hence, we obtain $E \vdash (\forall X)C\sigma$, which means that we can also apply the *Replacement*$_2$ rule by using just the equations in E to establish the condition C, and obtain $E \vdash (\forall X)t''\sigma : s$. The conclusion follows from $t' = t''\sigma$.
  - if the membership μ is $(\forall X)t : s$, then, there exists a kind-preserving substitution $\sigma : X \mapsto T_\Sigma(X)$ such that $t' = t\sigma$. We know by hypothesis that $E \vdash (\forall X)t : s$, and we can apply Lemma 8 to obtain $E \vdash (\forall X)t\sigma : s$, i.e., $E \vdash (\forall X)t' : s$. □

The next lemma characterises the entailment $\vdash_\mu$ (Definition 5) in the particular case of simply inductive sorts (Definition 8). Remember that if a sort s is simply inductive, then for any defining assertion $\mu \in \mathcal{D}(s)$, the condition C_μ can be decomposed as $C_\mu = C_{\neg s,\mu} \wedge C_{s,\mu}$, where $C_{s,\mu}$ is either *true* or a membership $(\forall X)t'_\mu : s$ for some term t'_μ.

Lemma 10 *Consider a* MEL *theory $\mathcal{M} = (\Sigma, E)$, a simply inductive sort s of $\mathcal{M}$ such that all memberships $\mu \in \mathcal{D}(s)$ are inductive, and a membership $(\mu)\,(\forall X)t_\mu : s$ if $C_\mu \in \mathcal{D}(s)$, where $C_\mu = C_{\neg s,\mu} \wedge C_{s,\mu}$ and $C_{s,\mu} = t'_\mu : s$. Then, for $t, t' \in T_\Sigma(Y)$: $E \cup \{(\forall Y)t : s\} \vdash_\mu (\forall Y)t' : s$ iff there exists a kind-preserving substitution $\sigma : X \mapsto T_\Sigma(Y)$ such that*

1. $E \vdash (\forall Y)t = t'_\mu\sigma$
2. $E \vdash (\forall Y)C_{\neg s,\mu}\sigma$
3. $E \vdash (\forall Y)t' = t_\mu\sigma$. □

Proof. ($\Rightarrow$) By complete induction on the length n of the derivation of the entailment $E \cup \{(\forall Y)t : s\} \vdash_\mu (\forall Y)t' : s$. The last step can be either an application of a *Membership* rule or a *Replacement*$_2$ rule - the *Membership'* rule is an instance of *Replacement*$_2$.

- if the last step is an application of the *Membership* rule, then there exists a term $\tilde{t} \in T_\Sigma(Y)$ such that the entailments $E \cup \{(\forall Y)t : s\} \vdash (\forall Y)t' = \tilde{t}$ and $E \cup \{(\forall Y)t : s\} \vdash (\forall Y)\tilde{t} : s$ are provable in strictly fewer than n steps. The induction hypothesis applied to $E \cup \{(\forall Y)t : s\} \vdash (\forall Y)\tilde{t} : s$ gives us

  1. $E \vdash (\forall Y)t = t'_\mu\sigma$
  2. $E \vdash (\forall Y)C_{\neg s,\mu}\sigma$
  3. $E \vdash (\forall Y)\tilde{t} = t_\mu\sigma$.

  This is almost the expected conclusion, except for the third item, which should be $E \vdash (\forall Y)t' = t_\mu\sigma$. To obtain this entailment, we use Lemma 7 with $E \cup \{(\forall Y)t : s\} \vdash (\forall Y)t' = \tilde{t}$ and obtain $E \vdash (\forall Y)t' = \tilde{t}$, which, by using the *Symmetry* and the *Transitivity* rules, implies $E \vdash (\forall Y)t' = t_\mu\sigma$, and the proof is done in this case.

- If the last step is an application of the *Replacement*$_2$ rule, then, by Definition 5 of $\vdash_\mu$, the rule is used with *the* membership (μ) $(\forall X)t_\mu : s$ *if* C_μ in the hypothesis of our lemma, and then there exists a kind-preserving substitution $\sigma : X \mapsto T_\Sigma(Y)$ such that (a) $E \cup \{(\forall t : s\} \vdash (\forall Y)C_\mu\sigma$ and (b) $t' = t_\mu\sigma$; (b) implies, *a fortiori*, $E \vdash (\forall Y)t' = t_\mu\sigma$, i.e., the third item in the conclusion. Next, (a) implies

  - $E \cup \{(\forall Y)t : s\} \vdash (\forall Y)C_{\neg s,\mu}\sigma$. Now, just like $C_{\neg s,\mu}$, $C_{\neg s,\mu}\sigma$ is a conjunction of equations and of memberships to sorts $s' \neq s$. Hence, we can apply Lemma 7 to all of them, and obtain $E \vdash (\forall Y)C_{\neg s,\mu}\sigma$, which proves the second item in the conclusion.
  - $E \cup \{(\forall Y)t : s\} \vdash C_{s,\mu}\sigma$, i.e., $E \cup \{(\forall Y)t : s\} \vdash (\forall Y)t'_\mu\sigma : s$. Using Corollary 2 we obtain $E \vdash (\forall Y)t = t'_\mu\sigma$. This proves also proves the first item in the conclusion and the ($\Rightarrow$) implication[10].

($\Leftarrow$) Using the *Membership'* rule, $E \cup \{(\forall Y)t : s\} \vdash (\forall Y)t : s$, which, combined with the hypothesis at item 1: $E \vdash (\forall Y)t = t'_\mu\sigma$ and the *Membership* rule, gives $E \cup \{(\forall Y)t : s\} \vdash (\forall Y)t'_\mu : s$, i.e., $E \cup \{(\forall Y)t : s\} \vdash (\forall Y)C_{s,\mu}$. Together with the hypothesis at item 2: $E \cup \{(\forall Y)t : s\} \vdash (\forall Y)C_{\neg s,\mu}$ we then get $E \cup \{(\forall Y)t : s\} \vdash (\forall Y)C_\mu$. We can now apply the *Replacement*$_2$ rule with the membership (μ) and substitution σ from the hypothesis and get $E \cup \{(\forall Y)t : s\} \vdash_\mu (\forall Y)t_\mu\sigma : s$ (we have used the *Replacement*$_2$ rule with the

[10] Note that Corollary 2 could be applied because its hypothesis - that there are no instances of the *Replacement*$_2$ rule used with memberships in $\mathcal{D}(s)$ in the derivation of $E \cup \{(\forall Y)t : s\} \vdash C_{s,\mu}\sigma$ - is satisfied. And the hypothesis is satisfied because, otherwise, we would have strictly more than one instance of the *Replacement*$_2$ rule used with memberships in $\mathcal{D}(s)$ in the derivation of $E \cup \{(\forall Y)t : s\} \vdash_\mu (\forall Y)t' : s$, in contradiction with Definition 5.

membership (μ) exactly once as required by the definition of $\vdash_\mu$). Finally, the last entailment together with the hypothesis at item 3: $E \vdash (\forall Y)t' = t_\mu\sigma$ and the *Membership* rule give us $E \cup \{(\forall Y)t : s\} \vdash_\mu (\forall Y)t' : s$, and the proof is done. □

The main result from Section 2.3 is:

Proposition 3 *Consider a* MEL *theory* $\mathcal{M} = (\Sigma, E)$*, and let* s *be simply inductive sort of* $\mathcal{M}$ *such that all the defining assertions of* s *in* E *are inductive. Then, for all* $t, t' \in T_\Sigma(X)$*:* $E \cup \{(\forall X)t : s\} \vdash (\forall X)t' : s$ *iff either* $E \vdash (\forall Y)t = t'$*, or there exists a sequence of memberships* $\mu_1, \ldots, \mu_n \in E \cap D(s)$ *(*$n \geq 1$*) and a sequence of terms* $t_i \in T_\Sigma(X)$ *for* $i \in [0, n-1]$ *such that* $t_0 = t$*,* $t_n = t'$*, and* $E \cup \{(\forall X)t_i : s\} \vdash_{\mu_{i+1}} (\forall X)t_{i+1} : s$ *for* $i \in [0, n-1]$*.* □

Proof. ($\Rightarrow$) Consider the (possibly empty) sequence of defining memberships for the sort s: $\mu_1, \ldots, \mu_n \in E \cap D(s)$ ($n \geq 0$) that are used, in this order, with the *Replacement*$_2$ rule in the derivation of the entailment $E \cup \{(\forall X)t : s\} \vdash (\forall X)t' : s$. The proof goes by simple induction on n. For the base case $n = 0$: in this case, the *Replacement*$_2$ rule is not used with memberships in $E \cap D(s)$, hence, by Corollary 2, $E \vdash (\forall Y)t = t'$. For the induction step, let $\mu_1, \ldots, \mu_n \in E \cap D(s)$ be the sequence of defining memberships for the sort s that occur, in this order, in the derivation of the entailment $E \cup \{(\forall X)t : s\} \vdash (\forall X)t' : s$. Without restricting the generality, we can assume that there is no shorter sequence of memberships than $\mu_1, \ldots, \mu_n$ that allows to derive our entailment.

- If $n = 1$ then $E \cup \{(\forall X)t : s\} \vdash_{\mu_1} (\forall X)t' : s$, i.e., the expected conclusion.
- If $n > 1$, then μ_n is the last membership used with the *Replacement*$_2$ rule in the derivation of the entailment $E \cup \{(\forall t : s\} \vdash (\forall X)t' : s$. Since all the defining assertions in $E \cap D(s)$ are inductive, they cannot *by themselves* derive anything - there must be a *root* of the derivation process, i.e., a term $t'' \in T_\Sigma(X)$ such that $E \cup \{(\forall X)t'' : s\} \vdash_{\mu_n} (\forall X)t' : s$. On the other hand, $(\forall X)t'' : s$ does not come "from heaven" - it must have been deduced from $E \cup \{(\forall X)t : s\}$, since these are the only available hypotheses. Hence, $E \cup \{(\forall X)t : s\} \vdash (\forall X)t'' : s$, and, by the minimality of the sequence of membership $\mu_1, \ldots, \mu_n$ in the derivation of $E \cup \{(\forall X)t : s\} \vdash (\forall X)t' : s$, the derivation of $E \cup \{(\forall X)t : s\} \vdash (\forall X)t'' : s$ uses $n-1$ memberships $\mu'_1, \ldots \mu'_{n-1}$ and the conclusion follows by the induction hypothesis.

($\Leftarrow$) If $E \vdash t = t'$ the proof is done. Otherwise, there exists a sequence of memberships $\mu_1, \ldots, \mu_n \in E \cap D(s)$ ($n \geq 1$) and a sequence of terms $t_i \in T_\Sigma(X)$ for $i \in [0, n-1]$ such that $t_0 = t$, $t_n = t'$, and $E \cup \{(\forall X)t_i : s\} \vdash_{\mu_{i+1}} (\forall X)t_{i+1} : s$ for $i \in [0, n-1]$, and we prove by induction on $n \geq 1$ that $E \cup \{(\forall X)t : s\} \vdash (\forall X)t' : s$. The base case $n = 1$ is trivial. We assume the statement is true for n and prove it for $n + 1$. The statement for n gives by induction hypothesis $E \cup \{(\forall X)t : s\} \vdash (\forall X)t_n : s$. The final deduction $\vdash_{\mu_{n+1}}$ is $E \cup \{(\forall X)t_n : s\} \vdash_{\mu_{n+1}} (\forall X)t' : s$, hence, by monotonicity of deduction, $E \cup \{(\forall X)t : s\} \cup \{(\forall X)t_n : s\} \vdash (\forall X)t' : s$ and by Lemma 9, $E \cup \{(\forall X)t : s\} \vdash (\forall X)t' : s$, and the proof is done. □

Proofs of results from Section 3.2

The main result is Proposition 4, which is similar to Proposition 3 above but deals with deduction using rewrite rules. We shall again need some intermediary results. The first one says that the *Replacement* rule is the one actually performing rewrites; without it, only equalities can be derived.

Lemma 11 *Consider a* RL *theory* $\mathcal{R} = (\Sigma, E, R)$ *and two terms* $t, t' \in T_\Sigma(X)$ *such that there exists a derivation of the entailment* $\mathcal{R} \vdash (\forall X)t \hookrightarrow t'$ *that does not use the* Replacement *rule. Then,* $E \vdash (\forall X)t = t'$.

Proof. By complete induction on the length $n \geq 1$ of the derivation of the entailment $\mathcal{R} \vdash (\forall X)t \hookrightarrow t'$. The last step of the derivation is either

- *Reflexivity*: then, the derivation can be decomposed into a prefix $\mathcal{R} \vdash (\forall X)t \hookrightarrow t'$ of length $n-1$, and the last reflexivity step $\mathcal{R} \vdash (\forall X)t' \hookrightarrow t'$. The conclusion follows immediately using the induction hypothesis.
- *Transitivity*: the derivation can be decomposed into two fragments of length $< n$ each: $\mathcal{R} \vdash (\forall X)t \hookrightarrow t''$ and $\mathcal{R} \vdash (\forall X)t'' \hookrightarrow t'$ for some term $t'' \in T_\Sigma(X)$. The conclusion follows using the induction hypothesis and transitivity of equality in MEL.
- *Equality*: then, there is a derivation of length n: $\mathcal{R} \vdash (\forall X)u \hookrightarrow u'$ such that $E \vdash (\forall X)t = u$ and $E \vdash (\forall X)u' = t'$. The conclusion follows using the induction hypothesis and transitivity of equality in MEL.

These are the only possibilities, since, by hypothesis, we have excluded the *Congruence* and *Replacement* rules, and the proof is done. □

We now characterise the top-level one-step entailment relation (Definition 12). This lemma will be used in proving a result from Section 4.1.

Lemma 12 *Let* $\rho : (\forall X)l \to r$ *if* C *be a rule of a* RL *theory* $\mathcal{R} = (\Sigma, E, R)$. *Then,* $\mathcal{R} \vdash_\rho (\forall Y)t \hookrightarrow t'$ *iff there exists a kind-preserving substitution* $\sigma : X \mapsto T_\Sigma(Y)$ *such that:*

1. $E \vdash (\forall Y)t = l\sigma$
2. $E \vdash (\forall Y)C\sigma$
3. $E \vdash (\forall Y)t' = r\sigma$. □

Proof. ($\Rightarrow$) By complete induction on the length $n \geq 1$ of the derivation of $\mathcal{R} \vdash_\rho (\forall Y)t \hookrightarrow t'$. The last step in the derivation may be

1. *Reflexivity*: in this case, the derivation can be decomposed into a prefix $\mathcal{R} \vdash_\rho (\forall Y)t \hookrightarrow t'$ of length n, and the last reflexivity step $\mathcal{R} \vdash (\forall Y)t' \hookrightarrow t'$. The conclusion follows immediately using the induction hypothesis.

2. *Transitivity*: the derivation can be decomposed into two fragments of length $< n$ each: $\mathcal{R} \vdash (\forall Y)t \hookrightarrow t''$ and $\mathcal{R} \vdash (\forall Y)t'' \hookrightarrow t'$ for some term $t'' \in T_\Sigma(Y)$. Now, the (unique) application of the replacement rule occurs either in the first or the second of these two fragments. If the *Replacement* occurs in the first one, $\mathcal{R} \vdash (\forall Y)t \hookrightarrow t''$, then we have $\mathcal{R} \vdash_\rho (\forall Y)t \hookrightarrow t''$, and using the induction hypothesis we obtain

   (a) $E \vdash (\forall Y)t = l\sigma$

   (b) $E \vdash (\forall Y)C\sigma$

   (c) $E \vdash (\forall Y)t'' = r\sigma$. □

   Hence, the first and second item of the conclusion are proved. For the third one, note that, since the *Replacement* rule is not used in the proof of $\mathcal{R} \vdash (\forall Y)t'' \hookrightarrow t'$, we obtain using Lemma 11 that $E \vdash (\forall Y)t'' = t'$, and the third item in the conclusion follows by transitivity of equality in MEL. The situation where the replacement occurs in $\mathcal{R} \vdash (\forall Y)t'' \hookrightarrow t'$ is similar.

3. *Equality*: there is a derivation of length n: $\mathcal{R} \vdash (\forall Y)u \hookrightarrow u'$ such that $E \vdash (\forall Y)t = u$ and $E \vdash (\forall Y)u' = t'$. The conclusion follows using the induction hypothesis and transitivity of equality in MEL.

4. *Replacement*: the derivation can be decomposed into a prefix $\mathcal{R} \vdash (\forall Y)t \hookrightarrow t''$ of length n, which does not involve the *Replacement* rule, and the last *Replacement* step using the rule ρ: $\mathcal{R} \vdash_\rho (\forall Y)t'' \hookrightarrow t'$. For the initial fragment we obtain using Lemma 11 that $E \vdash (\forall Y)t = t''$, and for the last step, we obtain from Definition 9 that there exists a kind-preserving substitution $\sigma : X \mapsto T_\Sigma(Y)$ such that:

   - $l\sigma = t''$
   - $E \vdash (\forall Y)C\sigma$
   - $t' = r\sigma$.

   The conclusion follows by the *Reflexivity* and *Transitivity* rules of MEL.

Since no *Congruence* rules are allowed (cf. Definition 12), we have exhausted all possibilities; this concludes the ($\Rightarrow$) direction of the proof.

($\Leftarrow$) Using Definitions 9 and 12 we obtain using the second item in the hypothesis that $\mathcal{R} \vdash_\rho (\forall Y)l\sigma = r\sigma$. Then, using the first and third items and the *Equality* rule of RL we obtain $\mathcal{R} \vdash_\rho (\forall Y)t \hookrightarrow t'$, and the proof is done. □

Proposition 4 *Consider a* RL *theory* $\mathcal{R} = (\Sigma, E, R)$. *Then, for all* $t, t' \in T_\Sigma(X)$: $\mathcal{R} \vdash (\forall X)t \hookrightarrow t'$ *iff either* $E \vdash (\forall X)t = t'$, *or there exists a sequence of rules* $\rho_1, \ldots, \rho_n \in R$ ($n \geq 1$) *and terms* $t_i \in T_\Sigma(X)$, $i \in [0, n-1]$ *such that* $t_0 = t$, $t_n = t'$, *and* $\mathcal{R} \vdash_{\rho_i} (\forall X)t_i \hookrightarrow t_{i+1}$ *for* $i \in [0, n-1]$. □

Proof. ($\Rightarrow$) is proved by routine induction on the number of rewrite rules used in $\mathcal{R} \vdash (\forall X)t \hookrightarrow t'$, and ($\Leftarrow$) is proved by routine induction on n. □

Proofs of results from Section 4.1

The main result is Proposition 6. We shall use the following simple corollary to Lemma 10 (N.B. remember also Definition 16 of the membership $\mu(\rho)$ associated to a rewrite rule ρ).

Corollary 3 *For all $t, t' \in T_\Sigma(Y)$: $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall t : Reachable\} \vdash_{\mu(\rho)} \{(\forall Y)t' : Reachable\}$ iff there exists a kind-preserving substitution $\sigma : X \mapsto T_\Sigma(Y)$ such that all the following statements hold:*

1. $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash (\forall Y)t = l\sigma$;
2. $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash (\forall Y)l\sigma : Reachable$;
3. $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash (\forall Y)C\sigma$;
4. $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash (\forall Y)t' = r\sigma$. □

Proposition 5 With the notations of Definition 17, for all $t, t' \in T_\Sigma(Y)$: $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash_{\mu(\rho)} (\forall Y)t' : Reachable$ *iff* $\mathcal{R} \vdash_\rho (\forall Y)t \hookrightarrow t'$. □

Proof. ($\Rightarrow$) Assume $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash_{\mu(\rho)} (\forall Y)t' : Reachable$. Using Corollary 3, we obtain that there exists a kind-preserving substitution $\sigma : X \mapsto T_\Sigma(Y)$ such that

- $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash (\forall Y)t = l\sigma$; since *Reachable* is a simply inductive sort (cf. Definition 8), memberships to *Reachable* cannot contribute to the proof of anything except other memberships to *Reachable* (cf. Lemma 7), hence, (a) $E \vdash (\forall Y)t = l\sigma$;
- $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash (\forall Y)C\sigma$; by the same observation as above, $E \vdash (\forall Y)C\sigma$; using (a) above and Lemma 12, we get (b) $\mathcal{R} \vdash_\rho (\forall Y)t \hookrightarrow r\sigma$;
- $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash (\forall Y)t' = r\sigma$; by the same observation as above, $E \vdash (\forall Y)t' = r\sigma$; hence, using (b) above, $\mathcal{R} \vdash_\rho (\forall X)t \hookrightarrow t'$, which is the conclusion of the ($\Rightarrow$) direction of the proof.

($\Leftarrow$) Assume $\mathcal{R} \vdash_\rho (\forall Y)t \hookrightarrow t'$. By Lemma 12, this means that there exists a kind-preserving substitution $\sigma : X \mapsto T_\Sigma(Y)$ such that

- $E \vdash (\forall Y)t = l\sigma$, hence, *a fortiori*, (d) $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash (\forall Y)t = l\sigma$; then, using the *Membership* and *Membership'* rules of MEL (cf. Definitions 1 and 3), we obtain (e) $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash (\forall Y)l\sigma : Reachable$;
- $E \vdash (\forall Y)C\sigma$, hence, *a fortiori*, $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash (\forall Y)C\sigma$
- $E \vdash (\forall Y)t' = r\sigma$, hence, *a fortiori*, (g) $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash (\forall Y)t' = r\sigma$.

From (d), (e), (f), and(g) we obtain, using Corollary 3, that $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall Y)t : Reachable\} \vdash_{\mu(\rho)} (\forall Y)t' : Reachable$, and the proof is done. □

Proposition 6 *Let $\mathcal{R} = (\Sigma, E, R)$ be an admissible* RL *theory and e an atomic state predicate of $\mathcal{R}$, i.e., either a membership $t : s$ or an equation $t = t'$, with $t, t' \in T_\Sigma(X)$. Then, $E \vdash (\forall X)e$ iff $\mathcal{M}(\mathcal{R}, t_0) \vdash (\forall X)e$.* □

Proof. First, remember that $\mathcal{M}(\mathcal{R}, t_0) \vdash (\forall X)e$ is merely a notation for $E_{\mathcal{M}(\mathcal{R})} \cup \{(\forall X)t_0 : \mathit{Reachable}\} \vdash (\forall X)e$, and, by Definition 17, $E_{\mathcal{M}(\mathcal{R})} = E \cup \{\mu(\rho) | \rho \in R\}$. Hence, we have to prove

$$\begin{aligned} E \quad &\vdash (\forall X)e \iff \\ E \quad &\cup \{\mu(\rho) | \rho \in R\} \cup \{(\forall X)t_0 : \mathit{Reachable}\} \vdash (\forall X)e \end{aligned}$$

The ($\Rightarrow$) implication is a consequence of the monotonicity of deduction in $\vdash$; and the ($\Leftarrow$) implication is a direct consequence of Lemma 7. Note that for ($\Leftarrow$) it is important that the *Reachable* sort is simply inductive. □