Invariant-preserved Transformation of State Machines from Equations into Rewrite Rules

Min Zhang, Kazuhiro Ogata
Research Center for Software Verification & Graduate School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
Email: {zhangmin,ogata}@jaist.ac.jp

***Abstract*—A state machine can be specified as either an equational theory or a rewrite theory in algebraic approaches. The former is used for theorem proving, and the latter for model checking. We have proposed an approach to transform a class of equational theories into rewrite theories in order to use them in the combination of the two verification techniques. This paper shows the correctness of the transformation with respect to its preservation of invariant properties. Invariant-preservation guarantees that a counterexample found by model checking a generated rewrite theory is also a counterexample of the same invariant in the original equational theory, which provides the theoretical support to the utilization of the transformation in combination of theorem proving and model checking.**

***Keywords*-Rewrite theory, equational theory, transformation, invariant, state machine, formal verification**

I. Introduction

Equational theories and rewrite theories are two main algebraic-based approaches to formalizing state machines [1], [2]. They are used to verify or falsify computer systems' properties in different techniques, e.g., the former are used for theorem proving [3], [1], and the latter for (bounded) model checking [4], [5]. In our earlier work [6], we proposed an approach to transforming a class of equational theories that specify state machines in CafeOBJ language [7] into rewrite theories in Maude language [4]. Generated Maude specifications are used by model checking to find counterexamples of invariant properties that are being verified by theorem proving in original equational theories [6]. However, when a counterexample is found for an invariant property by model checking, could we say the counterexample is also a valid one in the original equational theory? When no counterexamples are found, could we say the invariant property holds in the original theory?

To answer the above two questions, we show in this paper that the transformation preserves invariant properties, in that an invariant property holds in the generated rewrite theory if it holds in the original equational theory. Especially, the other direction also holds under the condition that each state in the state machine represented by the equational theory being translated has a finite number of successors. We call it invariant-preservation of the transformation.

Invariant-preservation property is fundamental to the model checking result of generated rewrite theories. It guarantees that an invariant must not hold in an original equational theory once a counterexample is found in the generated rewrite theory. Therefore, we can falsify an invariant by choosing a reasonable subset of states to model check. If no counterexamples are found, we then resort to theorem proving to prove it.

II. State machine and its Algebraic-based Models

A. State machine

State machine is used to model computer systems. A state machine is a triple $\mathscr{M} \triangleq \langle \mathbf{U}, \mathbf{I}, \mathbf{T} \rangle$, where $\mathbf{U}$ is a set of *states*, $\mathbf{I}$ is a set of *initial states* s.t. $\mathbf{I} \subseteq \mathbf{U}$, and $\mathbf{T}$ is a binary relation over $\mathbf{U}$. For each $u, u' \in \mathbf{U}$, if $(u, u') \in \mathbf{T}$, it denotes that there is a transition from u to u'. u' is a successor state of u.

Definition 1 (Reachable state): The set $\mathbf{R}_{\mathscr{M}}$ of reachable states of $\mathscr{M}$ is inductively defined as follows:

- for each $u \in \mathbf{I}_{\mathscr{M}}$, $u \in \mathbf{R}_{\mathscr{M}}$;
- for each $u, u' \in \mathbf{U}_{\mathscr{M}}$ s.t. $(u, u') \in \mathbf{T}_{\mathscr{M}}$ and $u \in \mathbf{R}_{\mathscr{M}}$, $u' \in \mathbf{R}_{\mathscr{M}}$.

Definition 2 (Invariants): Given a state machine $\mathscr{M}$, a state predicate $\rho : \mathbf{U} \rightarrow \mathcal{B}$ is an invariant w.r.t. $\mathscr{M}$ if $\forall u \in \mathbf{R}_{\mathscr{M}}.\rho(u)$, where $\mathcal{B} = \{true, false\}$.

B. Observational Transition Systems (OTSs)

OTSs describe state machines as equational theories [8]. Let Υ denote a universal state space, and D with a subscript denote a set of basic data used in OTSs such as natural numbers, integers, Boolean values, etc.

Definition 3 (OTSs): An OTS $\mathcal{S}$ is a triple $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$:

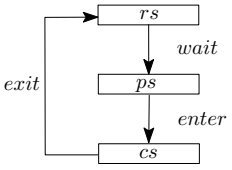
- $\mathcal{O}$: A set of observers. Each observer is a function $o : \Upsilon \times D_{o1} \times \ldots \times D_{om} \rightarrow D_o$. Two states υ_1, υ_2 are equal (denoted by $\upsilon_1 =_{\mathcal{S}} \upsilon_2$) if each observer returns the same value with the same arguments from the two states.
- $\mathcal{I}$: A set of initial states s.t. $\mathcal{I} \subseteq \Upsilon$.
- $\mathcal{T}$: A set of transitions. Each transition is a function $t : \Upsilon \times D_{t1} \times \ldots \times D_{tn} \rightarrow \Upsilon$. Each t preserves the equivalence between two states in that if $\upsilon_1 =_{\mathcal{S}} \upsilon_2$, then for each $y_i (i = 1, \ldots, n)$ in D_{ti}, $t(\upsilon_1, y_1, \ldots, y_n) =_{\mathcal{S}} t(\upsilon_2, y_1, \ldots, y_n)$. Each t has an effective condition $c\text{-}t : \Upsilon \times D_{t1} \times \ldots \times D_{tn} \rightarrow \mathcal{B}$, s.t. for any state υ if $\neg c\text{-}t(\upsilon, y_1, \ldots, y_n)$, $t(\upsilon, y_1, \ldots, y_n) =_{\mathcal{S}} \upsilon$.

An OTS is specified as an equational theory. Initial states are defined by a set of equations. Each equation corresponds to an observer $o : \Upsilon \times D_{o1} \times \ldots \times D_{om} \rightarrow D_o$, such that for each υ_0 in $\mathcal{I}$ and each $x_j (j = 1, \ldots, m)$ in D_{oj}:

$$o(\upsilon_0, x_1, \ldots, x_m) = f_o(\upsilon_0, x_1, \ldots, x_m) \quad (1)$$

Loop remainder section
rs: **repeat until** $locked = false$;
es: $locked := true$;
critical section
cs: $locked := false$;

(a). Pseudo code of the flawed mutex protocol



(b). The behavior of each process

$\mathcal{S}_{\text{FMP}} \triangleq \langle \mathcal{O}_{\text{FMP}}, \mathcal{I}_{\text{FMP}}, \mathcal{T}_{\text{FMP}} \rangle$, such that:
$\mathcal{O}_{\text{FMP}} \triangleq \{pc : \Upsilon \times P \to \mathcal{L}, locked : \Upsilon \to \mathcal{B}\}$
$\mathcal{I}_{\text{FMP}} \triangleq \{\upsilon_0 | \forall p \in P. pc(\upsilon_0, p) = rs \wedge locked(\upsilon_0) = false\}$
$\mathcal{T}_{\text{FMP}} \triangleq \{wait : \Upsilon \times P \to \Upsilon, enter : \Upsilon \times P \to \Upsilon, exit : \Upsilon \times P \to \Upsilon\}$
For each p, p' in P, and υ in Υ, there are:

- $c\text{-}wait(\upsilon, p) \triangleq pc(\upsilon, p) = rs \wedge locked(\upsilon) = false$. If $c\text{-}wait(\upsilon, p)$ holds:

$$pc(wait(\upsilon, p), p') = \begin{cases} es & \text{if } p = p' \\ pc(\upsilon, p') & \text{otherwise} \end{cases} \quad (1)$$

$$locked(wait(\upsilon, p)) = locked(\upsilon) \quad (2)$$

- $c\text{-}enter(\upsilon, p) \triangleq pc(\upsilon, p) = es$. If $c\text{-}enter(\upsilon, p)$ holds:

$$pc(enter(\upsilon, p), p') = \begin{cases} cs & \text{if } p = p' \\ pc(\upsilon, p') & \text{otherwise} \end{cases} \quad (3)$$

$$locked(enter(\upsilon, p)) = true \quad (4)$$

- $c\text{-}exit(\upsilon, p) \triangleq pc(\upsilon, p) = cs$. If $c\text{-}exit(\upsilon, p)$ holds:

$$pc(exit(\upsilon, p), p') = \begin{cases} rs & \text{if } p = p' \\ pc(\upsilon, p') & \text{otherwise} \end{cases} \quad (5)$$

$$locked(exit(\upsilon, p)) = false \quad (6)$$

(c). An OTS $\mathcal{S}_{\text{FMP}}$ of the protocl

Fig. 1. A flawed mutex protocol and its OTS model

where f_o is a function, returning a value in D_o from a given state. Transition operators are not allowed to occur in the function. The equation for an observer o w.r.t. a transition t is defined in the following form:

$$o(\upsilon', x_1, \ldots, x_m) = f'_o(\upsilon, y_1, \ldots, y_n, x_1, \ldots, x_m) \quad (2)$$

where $\upsilon' \triangleq t(\upsilon, y_1, \ldots, y_n)$, and f'_o returns a value in D_o from a given state with appropriate arguments. Equation 2 holds if $c\text{-}t(\upsilon, y_1, \ldots, y_n)$ is true. It represents how the values that are returned by o are changed by t w.r.t. $y_1, \ldots, y_n$.

Let us consider an OTS of a flawed mutex protocol. Pseudo code of the protocol is shown as Fig. 1(a). Initially, each process is at the remainder section (rs). The behaviors of each process is depicted by Fig. 1(b). A process waits to enter the critical section (cs) until *locked* becomes false. It sets *locked* true, and then enters the critical section. When leaving, it sets *locked* false. OTS $\mathcal{S}_{\text{FMP}}$ in Fig. 1(c) models the protocol with an arbitrary (probably infinite) number of processes. Let P be a set of processes, and $\mathcal{L}$ be a set $\{rs, es, cs\}$ of labels. There are two observers pc and $locked$. Given a state υ and a process p, $pc(\upsilon, p)$ denotes the program counter of p in υ, and $locked(\upsilon)$ denotes the value of variable $locked$. We declare three transitions $wait$, $enter$, and $exit$. Each has an effective condition. For instance, $c\text{-}wait(\upsilon, p)$ denotes if p can move to es from υ. If it is true, $wait(\upsilon, p)$ represents the state after p moves to es. The values returned by pc and $locked$ in $wait(\upsilon, p)$ are defined by equations (1) and (2).

A state machine of an OTS $\mathcal{S}$ is interpreted as $\mathscr{M}_{\mathcal{S}}$ with $\mathbf{U}_{\mathcal{S}} \triangleq \Upsilon$, $\mathbf{I}_{\mathcal{S}} \triangleq \mathcal{I}$ and $\mathbf{T}_{\mathcal{S}} \triangleq \{(\upsilon_1, \upsilon_2) | \upsilon_1, \upsilon_2 \in \mathbf{U}_{\mathcal{S}}, \exists t \in \mathcal{T}, \exists y_1 : D_{t1}, \ldots, \exists y_n : D_{tn}. \upsilon_2 =_{\mathcal{S}} t(\upsilon_1, y_1, \ldots, y_n) \wedge c\text{-}t(\upsilon_1, y_1, \ldots, y_n)\}$. Thus, each state $\upsilon \in \mathbf{U}_{\mathcal{S}}$ has a finite number of successors if each D_{ti} of any transition t is finite. We use the latter as an assumption in the proof of the invariant-preservation in Section IV.

C. Component-based Transition Systems (CTSs)

In CTSs, a state is represented by a set of components, which we call a *configuration*. This is inspired by the treatment of states based on an associative and commutative (AC) operation in rewriting logic [9]. Unlike [9], there are two kinds of components in CTSs, *observational* and *transitional* components. Each observational component represents a value in the state. For instance, $(locked : true)$ is an observational component, representing that the variable $locked$ is true in the flawed mutex protocol. Transitional component represents transition information. Suppose that there are two processes p_1 and p_2 in the protocol, a transitional component $enter((p_1\ p_2))$ represents two possible transition instances caused by processes entering the critical section. Let O and T denote the sets of observational and transitional components respectively. The operator $locked{:}\_ : \mathcal{B} \to \mathsf{O}$ and $enter : \mathcal{P}(P) \to \mathsf{T}$ are corresponding component constructors. $\mathcal{P}(S)$ denotes the power set of a set S by convention. Particularly, $\mathcal{P}(\mathsf{O} \cup \mathsf{T})$ is the power set of components in O and T. We call it a set $\mathcal{C}$ of configurations.

Transitions among states are formalized as rewrite rules, specifying how values are changed. For instance, $(locked : true) \Rightarrow (locked : false)$ specifies the transition from those states where $locked$ is true to those where $locked$ is false, while other values are not changed. In rewrite rules, we only need to consider those components involved in transitions. A component is involved if it is changed, used as condition, or referenced by other changed components. A subset of components in a configuration is called a segment.

Definition 4 (CTSs): A CTS $\mathscr{S}$ is a four-tuple $\langle \mathsf{C}_o, \mathsf{C}_t, \psi, \mathsf{R} \rangle$, where:

- C_o : a set of observational component constructors e.g. $o[\_, \ldots, \_]{:}\_ : D_{o1} \times \ldots \times D_{om} \times D_o \to \mathsf{O}$;
- C_t : a set of transitional component constructors e.g. t :

$\mathsf{C}_o \triangleq \{locked\text{:}\_ : \mathcal{B} \to \mathsf{O}, pc[\_]\text{:}\_ : P \times \mathcal{L} \to \mathsf{O}\}$
$\mathsf{C}_t \triangleq \{wait : \mathcal{P}(P) \to \mathsf{T}, enter : \mathcal{P}(P) \to \mathsf{T}, exit : \mathcal{P}(P) \to \mathsf{T}\}$
$\psi \triangleq \psi : \mathcal{P}(P) \to \mathcal{C}$ such that for each $ps \in \mathcal{P}(P)$:
$\psi(ps) \triangleq wait(ps)\ enter(ps)\ exit(ps)\ (locked : false)\ mk\text{-}pc(ps)$
where, $mk\text{-}pc : \mathcal{P}(P) \to \mathcal{P}(\mathsf{O})$. For each p in P, ps in $\mathcal{P}(P)$:
$mk\text{-}pc(\emptyset) = \emptyset'$
$mk\text{-}pc(p\ ps) \triangleq (pc[p] : rs)\ mk\text{-}pc(ps)$
$\mathsf{R} \triangleq \{\forall p \in P.(locked : false)(pc[p] : rs) \Rightarrow (locked : false)(pc[p] : es),$
$\forall b \in B \forall p \in P.(locked : b)(pc[p] : es) \Rightarrow (locked : true)(pc[p] : cs),$
$\forall b \in B \forall p \in P.(locked : b)(pc[p] : cs) \Rightarrow (locked : false)(pc[p] : rs)\}$

Fig. 2. A CTS of the flawed mutual exclusion protocol

$\mathcal{P}(D_{t1}) \times \ldots \times \mathcal{P}(D_{tn}) \to \mathsf{T}$;
- ψ : an initial configuration generator e.g. $\psi : D_1 \times \ldots \times D_l \to \mathcal{C}$;
- R : a set of rewrite rules.

Fig. 2 shows a CTS of the flawed mutex protocol. Initial configurations are constructed by ψ, which takes a set of processes, and returns a configuration. Three rewrite rules in R represent three behaviors of each process. A configuration consists of three transitional components and $|P| + 1$ observational components. $|P|$ denotes the cardinality of P. Computationally, the successor is obtained by applying the first rule in R with p being instantiated by p_1. A segment $(locked : false)(pc[p_1] : rs)$ of the initial configuration is rewritten into $(locked : false)(pc[p_1] : es)$ with the rest unchanged. We repetitively apply the rules in R and can obtain a configuration containing $(pc[p_1] : cs)(pc[p_2] : cs)$, denoting both p_1, p_2 are at the critical section. A counterexample is found. Some tools are implemented to automate the procedure e.g., Maude's `search` command, LTL model checker [4].

Let c denote a configuration, δ be a subset of components in c. We call δ a segment of c, and the subset of the rest components of c except δ a context $\mathscr{C}$. The configuration c can be written as $\mathscr{C}[\delta]$.

Definition 5 ($\longrightarrow_r^1$): Given a CTS $\mathscr{S}$, for two configurations $c_1, c_2 \in \mathcal{C}$, $c_1 \longrightarrow_r^1 c_2$ if there is a rewrite rule $r \in \mathsf{R}$, a context $\mathscr{C}$ of configuration and a substitution σ s.t. $c_1 = \mathscr{C}[\sigma(L)], c_2 = \mathscr{C}[\sigma(R)]$, and $\sigma(C)$ is true.

We use r to denote a conditional rewrite rule $L \Rightarrow R$ *if* C, where L, R denotes configurations (or segments of configurations), and C denotes the condition. An unconditional rule can be considered as conditional with C always being true. $c_1 \longrightarrow_r^1 c_2$ means that c_1 can be rewritten by the rule r and c_2 is one instance obtained by applying r to c_1.

Definition 6 (Reachable configuration): The set $\mathcal{R}_\mathscr{S}$ of reachable configurations of a CTS $\mathscr{S}$ is recursively defined:

- $\mathcal{I}_\mathscr{S} \subseteq \mathcal{R}_\mathscr{S}$, where $\mathcal{I}_\mathscr{S} \triangleq \{c_0 | \exists d_1 \in D_1 \ldots \exists d_l \in D_l . c_0 = \psi(d_1, \ldots, d_n)\}$
- For any $c_1, c_2 \in \mathcal{C}$, if $c_1 \in \mathcal{R}_\mathscr{S}$ and there exists $r \in \mathsf{R}$ s.t. $c_1 \longrightarrow_r^1 c_2$, then $c_2 \in \mathcal{R}_\mathscr{S}$.

Given a configuration c, let $\lfloor c \rfloor$ denote the set of all the observational components in c. Then, we can interpret a CTS $\mathscr{S}$ as a state machine $\mathscr{M}_\mathscr{S} \triangleq \langle \mathbf{U}_\mathscr{S}, \mathbf{I}_\mathscr{S}, \mathbf{T}_\mathscr{S} \rangle$, where $\mathbf{U}_\mathscr{S} \triangleq \{\lfloor c \rfloor | c \in \mathcal{R}_\mathscr{S}\}$, $\mathbf{I}_\mathscr{S} \triangleq \{\lfloor c \rfloor | c \in \mathcal{I}_\mathscr{S}\}$, and $\mathbf{T}_\mathscr{S} \triangleq \{(\lfloor c_1 \rfloor, \lfloor c_2 \rfloor) | c_1, c_2 \in \mathcal{R}_\mathscr{S}, \exists r \in \mathsf{R}. c_1 \longrightarrow_r^1 c_2\}$.

III. Transformation from OTSs to CTSs

The basic idea of the transformation is to represent each value returned by an observer in $\mathcal{O}$ in OTS as an observational component, and to construct a rewrite rule for each transition in $\mathcal{T}$. We have proved that not all OTSs have such corresponding CTSs based on the undecidability of Post's Correspondence Problem (PCP) [6]. From pragmatic point of view, we define a subclass of OTSs, each of which can be transformed into a CTS in this way.

A. A subclass of OTSs

For each OTS in the subclass, there are two kinds of observers, which are in the form of either $o : \Upsilon \to D_o$ or $\hat{o} : \Upsilon \times D_p \to D_{\hat{o}}$, where D_p is a set of data. Moreover, each D_p of the second kind observers must be the same. By default, an observer without a hat denotes a first kind one, and the one with a hat denotes a second kind one. The equation declared for an observer o w.r.t a transition $t : \Upsilon \times D_{t1} \times \ldots \times D_{tn} \to \Upsilon$ must conform to Equation 3. The one for $\hat{o}$ must be in the form of Equation 4 or 5. As for Equation 5, D_p must be in the arity of t. We assume $D_{ti} (1 \leq i \leq n)$ is D_p.

$$o(t(\upsilon, y_1, \ldots, y_n)) = f_o(\upsilon, y_1, \ldots, y_n) \tag{3}$$

$$\forall x_p \in D_p . (\hat{o}(t(\upsilon, y_1, \ldots, y_n), x_p) = \hat{o}(\upsilon, x_p)) \tag{4}$$

$$\forall x_p, x_p' \in D_p . \hat{o}(t(\upsilon, y_1, \ldots, y_{i-1}, x_p', y_{i+1}, \ldots, y_n), x_p) = \begin{cases} f_{\hat{o}}(\upsilon, y_1, \ldots, y_{i-1}, x_p', y_{i+1}, \ldots, y_n) & \text{if } x_p = x_p' \\ \hat{o}(\upsilon, x_p) & \text{otherwise} \end{cases} \tag{5}$$

Equation 3 means the value that are observed by o is changed into $f_o(\upsilon, y_1, \ldots, y_n)$ by the transition from υ into $t(\upsilon, y_1, \ldots, y_n)$, and Equation 4 means no values observed by $\hat{o}$ are changed. Equation 5 means that the value observed by $\hat{o}$ w.r.t. x_p is changed into $f_{\hat{o}}(\upsilon, y_1, \ldots, y_n)$, and no other values are changed. The three equations guarantee that at most one value observed by each observer is changed by a transition, and it is decidable to compute which value is changed.

B. The transformation and optimization

An OTS in the subclass is transformed into an intermediate CTS. The CTS is optimized to a simpler one. The transformation consists of three steps as shown in Fig. 3, i.e., (a) to translate observers and transitions into their corresponding component constructors, (b) to define initial configuration generator, and (c) to translate equations into rewrite rules.

We assume there are $k_1 (k_1 \geq 0)$ and $k_2 (k_2 \geq 0)$ observers of the first and second kind respectively, and $k_3 (k_3 > 0)$ transitions in the OTS to be transformed. The transformation of observers and transitions are straightforward, as shown in Fig. 3(a). Note that a transitional component constructor takes the power set $\mathcal{P}(D_{ti})$ in its arity, if D_{ti} is in the arity of transition t. That is because a transitional component needs to represent all possible transition instances, like the components constructed by $wait$, $enter$, and $exit$ in Fig. 2.

$h_o : \mathcal{O} \to \mathsf{C}_o \quad h_t : \mathcal{T} \to \mathsf{C}_t$, where:

$h_o(obs) \triangleq$

$$\begin{cases} o\text{:\_} : D_o \to \mathsf{O} & \text{if } obs \equiv o : \Upsilon \to D_o \\ \hat{o}[\_]\text{:\_} : D_p \times D_{\hat{o}} \to \mathsf{O} & \text{if } obs \equiv \hat{o} : \Upsilon \times D_p \to D_{\hat{o}} \end{cases}$$

$h_t(tran) \triangleq t : \mathcal{P}(D_{t1}) \times \ldots \times \mathcal{P}(D_{tn}) \to \mathsf{T}$
if $tran \equiv t : \Upsilon \times D_{t1} \times \ldots \times D_{tn} \to \Upsilon$

(a). Transformation of observers and transitions

$\psi : \mathcal{P}(D_p) \times \mathcal{P}(D_1) \times \ldots \times \mathcal{P}(D_l) \to \mathcal{C}$, such that $\forall ps \in \mathcal{P}(D_p).\forall ys_1 \in \mathcal{P}(D_1) \ldots \forall ys_l \in \mathcal{P}(D_l).\psi(ps, ys_1, \ldots, ys_l) \triangleq t_1(ys_{11}, \ldots, ys_{1n_1}) \ldots t_{k_3}(ys_{k_3 1}, \ldots, ys_{k_3 n_{k_3}})(o_1\text{:}\ f_{o_1}(\upsilon_0)) \ldots\ (o_{k_1}\text{:}\ f_{o_{k_1}}(\upsilon_0))\ mk\text{-}\hat{o}_1(ps) \ldots mk\text{-}\hat{o}_{k_2}(ps)$

For each $mk\text{-}\hat{o}_i : \mathcal{P}(D_p) \to \mathcal{P}(\mathsf{O})$:

$mk\text{-}\hat{o}_i(\emptyset) \triangleq \emptyset'$

$mk\text{-}\hat{o}_i(p\ ps) \triangleq (\hat{o}_i[p] : f_{\hat{o}_i}(\upsilon_0, p))\ (mk\text{-}\hat{o}_i(ps))$

(b). Transformation of initial states

$$\varphi \left(\begin{array}{c} c\text{-}t(\upsilon, y_1, \ldots, y_n) = f_t(\upsilon, y_1, \ldots, y_n) \\ o_1(t(\upsilon, y_1, \ldots, y_n)) = f_{o_1}(\upsilon, y_1, \ldots, y_n) \\ \vdots \\ o_{k_1}(t(\upsilon, y_1, \ldots, y_n)) = f_{o_{k_1}}(\upsilon, y_1, \ldots, y_n) \\ \hat{o}_1(t(\upsilon, y_1, \ldots, y_{i-1}, x'_{p,1}, y_{i+1}, \ldots, y_n), x_p) = \begin{cases} f_{\hat{o}_1}(\upsilon, y_1, \ldots, y_{i-1}, x'_{p,1}, y_{i+1}, \ldots, y_n) & \text{if } x_p = x'_{p,1} \\ \hat{o}_1(\upsilon, x_p) & \text{otherwise} \end{cases} \\ \vdots \\ \hat{o}_{k_2}(t(\upsilon, y_1, \ldots, y_{i-1}, x'_{p,k_2}, y_{i+1}, \ldots, y_n), x_p) = \begin{cases} f_{\hat{o}_{k_2}}(\upsilon, y_1, \ldots, y_{i-1}, x'_{p,k_2}, y_{i+1}, \ldots, y_n) & \text{if } x_p = x'_{p,k_2} \\ \hat{o}_{k_2}(\upsilon, x_p) & \text{otherwise} \end{cases} \end{array} \right) \triangleq$$

$t(y_1\ ys_1, \ldots, y_n\ ys_n)(o_1 : z_1) \ldots (o_{k_1} : z_{k_1})(\hat{o}_1[x'_{p,1}] : z_{k_1+1}) \ldots (\hat{o}_{k_2}[x'_{p,k_2}] : z_{k_1+k_2}) \Rightarrow t(y_1\ ys_1, \ldots, y_t\ ys_n)(o_1 : f_{o_1}(\upsilon, y_1, \ldots, y_n)) \ldots (o_{k_1} : f_{o_{k_1}}(\upsilon, y_1, \ldots, y_n))(\hat{o}_1[x'_{p,1}] : f_{\hat{o}_1}(\upsilon, y_1, \ldots, y_n)) \ldots (\hat{o}_{k_2}[x'_{p,k_2}] : f_{\hat{o}_{k_2}}(\upsilon, y_1, \ldots, y_n))$ *if* $f_t(\upsilon, y_1, \ldots, y_n)\ [z_i \mapsto o_i(\upsilon), z_{k_1+j} \mapsto \hat{o}_j(\upsilon, x'_{p,j})]$, where $i = 1, \ldots, k_1$ and $j = 1, \ldots, k_2$.

(c). Transformation of equations

Fig. 3. Transformation from OTS to CTS

Next, we declare ψ to represent initial configurations, as shown in Fig. 3(b). For each $\mathcal{P}(D_i)(i = 1, \ldots, l)$, D_i must be in the arity of at least one transition, and for each D_{ti} in the arity of each t there is $\mathcal{P}(D_{ti})$ must be in the arity of ψ. In the definition of ψ, $(o_j : f_{o_j}(\upsilon_0))(j = 1, \ldots, k_1)$ corresponds to the value observed by o_j in initial states. Each $mk\text{-}\hat{o}_k(ps)(k = 1, \ldots, k_2)$ returns a set of observational components, each of which corresponds to a value observed by $\hat{o}_j$ with a process p in ps. Each $t_i(ys_{i1}, \ldots, ys_{in_i})(i = 1, \ldots, k_3)$ is a transitional component, where $ys_{in_k} \in \{ys_1, \ldots, ys_l\}(k = 1, \ldots, k_3)$.

Function φ takes a set of equations that are defined for a transition t and generates a rewrite rule, as shown in Fig. 3(c). In the transitional component $t(y_1\ ys_1, \ldots, y_n\ ys_n)$, y_i is an arbitrary element in D_{ti}, and ys_n is an arbitrary set in $\mathcal{P}(D_{ti})$. Thus, $(y_i\ ys_i)$ denotes a non-empty set in $\mathcal{P}(D_{ti})$. Since there is at most one value changed among the values returned by an observer, we construct a corresponding observational component for each value in the lefthand side e.g., $(\hat{o}_1[x'_{p,1}] : z_{k_1+1})$, and the component into which it is changed e.g., $(\hat{o}_1[x'_{p,1}] : f_{\hat{o}_1}(\upsilon, y_1, \ldots, y_n))$. For instance, the value observed by pc with a process p is changed into cs after p enters the critical section, according to the Equation (3) in Fig. 1(c). We construct a component $(pc[p] : z_p)$ where z_p is an arbitrary label in $\mathcal{L}$. The component is changed into $(pc[p] : cs)$. Thus, we put $(pc[p] : z_p)$ in the lefthand side of the rule for the transition $enter$, and $(pc[p] : cs)$ at the righthand side. Because $enter$ takes place only if pc of p is es, namely $z_p = es$ according to the effective condition of $enter$, we need to add it as the condition of the generated rewrite rule. In general case, we add the effective condition of a transition $t\ f_t(\upsilon, y_1, \ldots, y_n)$ to the rule generated for t.

1). Condition simplification g_c

$g_c((L \Rightarrow R\ if\ x = T)) \triangleq L[x \mapsto T] \Rightarrow R[x \mapsto T]$

$g_c((L \Rightarrow R\ if\ x = T \wedge C)) \triangleq L[x \mapsto T] \Rightarrow R[x \mapsto T]\ if\ C[x \mapsto T]$

where, x is a variable, and T is an *M-pattern* term and $x \notin \mathcal{V}(T)$

2). Variable elimination g_v

$g_v(r) \triangleq t((y_1\ ys_1), \ldots, (y_{i-1}\ ys_{i-1}), (y_{i+1}\ ys_{i+1}), \ldots, (y_n\ ys_n))\ L \Rightarrow t((y_1\ ys_1), \ldots, (y_{i-1}\ ys_{i-1}), (y_{i+1}\ ys_{i+1}), \ldots, (y_n\ ys_n))\ R\ if\ C$ if $y_{ti} \in \mathcal{V}(L)$, or $y_{ti} \notin \mathcal{V}(R) \cup \mathcal{V}(C)$

where, r is $t((y_1\ ys_1), \ldots, (y_i\ ys_i), \ldots, (y_n\ ys_n))\ L \Rightarrow t((y_1\ ys_1), \ldots, (y_i\ ys_i), \ldots, (y_n\ ys_n))\ R\ if\ C$.

3). Component elimination g_o

$g_o((o : z_i)L \Rightarrow (o : z_i)R\ if\ C) \triangleq L \Rightarrow R\ if\ C$ if $z_i \notin \mathcal{V}(R) \cup \mathcal{V}(C)$

$g_o((\hat{o}[x'_{p,i}] : z_i)L \Rightarrow (\hat{o}[x'_{p,i}] : z_i)R\ if\ C) \triangleq L \Rightarrow R\ if\ C$
if $z_i \notin \mathcal{V}(R) \cup \mathcal{V}(C)$, $x'_{p,i} \in \mathcal{V}(L)$ or $x'_{p,i} \notin \mathcal{V}(R) \cup \mathcal{V}(C)$

$g_o((t\ L \Rightarrow t\ R\ if\ C)) \triangleq L \Rightarrow R\ if\ C$

Fig. 4. Optimizations of CTSs

For example, φ takes the equations defined for $enter$ in $\mathcal{S}_{\text{FMP}}$, and generates a rewrite rule, i.e., for each p in P, ps in $\mathcal{P}(P)$, z_1 in $\mathcal{B}$, and z_2 in $\mathcal{L}$, $enter((p\ ps))(locked : z_1)(pc[p] : z_2) \Rightarrow enter((p\ ps))(locked : true)(pc[p] : cs)\ if\ z_2 = es$. The rule says that if p is at the location es it sets $locked$ true, and enters the critical section cs.

Optimizations of generated CTSs can be formalized by the three functions shown in Fig. 4. Function g_c simplifies the condition of a rewrite rule. T is an *M-pattern* term if for any well-formed substitution σ s.t. for each variable x in its

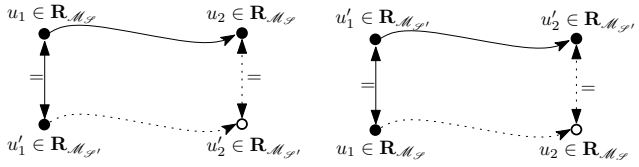


Fig. 5. Diagrams of Lemma 1 and Lemma 2

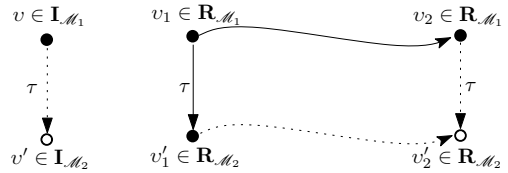


Fig. 6. A simulation τ from $\mathscr{M}_1$ to $\mathscr{M}_2$

domain the term $\sigma(x)$ is in canonical form w.r.t. the equations in an equational theory M, then $\sigma(T)$ is also in canonical form [4]. $\mathcal{V}(T)$ denotes the set of all variables in T. Function g_v removes unnecessary arguments from transitional component in a rewrite rule. An argument $(y_i\ ys_i)$ is unnecessary when y_i occurs in L, or does not occurs in R and C. Consequently, we need to change the arity of the constructor once an argument is removed. Function g_o removes unnecessary observational components from rewrite rules. An observational component $(o : z_i)$ is unnecessary in a rule if z_i is neither changed by the rule nor referenced by other components. If an observational component $(\hat{o}[x'_{p,i}] : z_i)$ is unnecessary, $x'_{p,i}$ must occur in L or not occur in R and C, besides the above condition.

For instance, the rule generated by φ for $enter$ can be simplified by g_c with z_2 in the lefthand side being replaced by es. We obtain an unconditional rule: $enter((p\ ps))(locked : z_1)(pc[p] : es) \Rightarrow enter((p\ ps))(locked : true)(pc[p] : cs)$. Further, g_v can be applied to the simplified rule because the argument $(p\ ps)$ is unnecessary and can be removed. Then, g_o is applied to the simplified one. We finally obtain the rule $(locked : z_1)(pc[p] : es) \Rightarrow (locked : true)(pc[p] : cs)$. It is semantically the same as the one in the CTS in Fig. 2.

IV. Invariant-preservation

Let $\mathcal{S}$ denote an OTS that conforms to constrains in Section III-A, $\mathscr{S}$ be the generated CTS before optimizations, and $\mathscr{S}'$ the optimized one of $\mathscr{S}$. We assume that each set D_{ti} taken by a transition t in $\mathcal{S}$ is finite. The basic idea is to prove that the optimization preserves reachable states of the state machines underlying the CTSs, and the transformation from $\mathcal{S}$ to $\mathscr{S}$ preserves invariant properties.

A. Reachable states preservation

First, we show that $\mathscr{M}_\mathscr{S}$ and $\mathscr{M}'_\mathscr{S}$ have the same sets of reachable states. It suffices to show that after each step of optimization, the state space is not changed. Thus, we consider $\mathscr{S}'$ as an optimized CTS of $\mathscr{S}$ by applying one optimization function to one rewrite rule in $\mathscr{S}$. We need two lemmas as shown in Fig. 5.

Lemma 1: For all $u_1, u_2 \in \mathbf{R}_{\mathscr{M}_\mathscr{S}}$, and for all $u'_1 \in \mathbf{R}_{\mathscr{M}_{\mathscr{S}'}}$ s.t. $(u_1, u_2) \in \mathbf{T}_{\mathscr{M}_\mathscr{S}}$ and $u'_1 = u_1$, there exists $u'_2 \in \mathbf{R}_{\mathscr{M}_{\mathscr{S}'}}$ s.t. $(u'_1, u'_2) \in \mathbf{T}_{\mathscr{M}_{\mathscr{S}'}}$ and $u'_2 = u_2$.

Proof: Since $u_1, u_2 \in \mathbf{R}_{\mathscr{M}_\mathscr{S}}$ and $(u_1, u_2) \in \mathbf{T}_{\mathscr{M}_\mathscr{S}}$, there exist $c_1, c_2 \in \mathcal{C}_\mathscr{S}$ and $r \in \mathcal{R}_\mathscr{S}$ s.t. $u_1 = \lfloor c_1 \rfloor, u_2 = \lfloor c_2 \rfloor$ and $c_1 \longrightarrow^1_r c_2$. We assume g_v is applied to r, i.e., $g_v(r) \in \mathcal{R}_{\mathscr{S}'}$. Let L_r, R_r, C_r are the terms at lefthand side, righthand side and condition. If $g_v(r) = r$, the result is straightforward. Otherwise, r satisfies the condition of g_v. There must exist a transitional component $\mathbf{t} \triangleq t(ds_1, \ldots, ds_{i-1}, ds_i, ds_{i+1}, \ldots, ds_n)$ in c_1, to which L_r can match. Let c'_1 be the configuration such that $\lfloor c_1 \rfloor = \lfloor c_2 \rfloor$, with ds_i removed from $\mathbf{t}$. Because $u'_1 = u_1$, $u'_1 = \lfloor c_1 \rfloor = \lfloor c'_1 \rfloor$. Then, $c_1 \longrightarrow^1_r c_2$ implies that there exists a substitution σ and a context $\mathscr{C}$ of configuration s.t. $c_1 = \mathscr{C}[L_r]$, $c_2 = \mathscr{C}[\sigma(R_r)]$and $\sigma(C)$ is true. Thus, $c'_1 = \mathscr{C}[\sigma(L_{g_v(r)})]$. Let c'_2 be $\mathscr{C}[\sigma(R_{g_v(r)})]$. $c'_2 \in \mathcal{C}_{\mathscr{S}'}$, $\lfloor c'_2 \rfloor = \lfloor c_2 \rfloor$, and $c'_1 \longrightarrow^1_{g_v(r)} c'_2$. Let u'_2 be $\lfloor c'_2 \rfloor$. $u'_2 \in \mathbf{R}_{\mathscr{M}_{\mathscr{S}'}}$, $(u'_1, u'_2) \in \mathbf{T}_{\mathscr{M}_{\mathscr{S}'}}$ and $u'_2 = u_2$.

The two other cases of g_c and g_o can be argued likewise, which are omitted from the paper. ■

Lemma 1 means that no transitions are lost after optimizations. We also need to show are transitions are added by optimizations, which is defined by Lemma 2.

Lemma 2: For all $u'_1, u'_2 \in \mathbf{R}_{\mathscr{M}_{\mathscr{S}'}}$, and for all $u_1 \in \mathbf{R}_{\mathscr{M}_\mathscr{S}}$ s.t. $(u'_1, u'_2) \in \mathbf{T}_{\mathscr{M}_{\mathscr{S}'}}$ and $u_1 = u'_1$, there exists $u_2 \in \mathbf{R}_{\mathscr{M}_\mathscr{S}}$ s.t. $(u_1, u_2) \in \mathbf{T}_{\mathscr{M}_\mathscr{S}}$ and $u_2 = u'_2$.

The proof of Lemma 2 is similar to the one of Lemma 1. Details of the proof can be referred to [10].

Theorem 1: Transitions in a CTS $\mathscr{S}$ are preserved in the its optimized one $\mathscr{S}'$.

It is equal to show $\mathbf{R}_{\mathscr{M}_\mathscr{S}} = \mathbf{R}_{\mathscr{M}_{\mathscr{S}'}}$. We can prove by induction on $\mathbf{R}_{\mathscr{M}_\mathscr{S}}$ and $\mathbf{R}_{\mathscr{M}_{\mathscr{S}'}}$. We omit the details.

Theorem 1 says that the state machine of an optimized CTS $\mathscr{S}'$ has the same reachable state space as the one of $\mathscr{S}$.

B. Simulation between $\mathscr{M}_\mathcal{S}$ and $\mathscr{M}_\mathscr{S}$

We show the simulation between the state machine $\mathscr{M}_\mathcal{S}$ of the original OTC $\mathcal{S}$ and the one $\mathscr{M}_\mathscr{S}$ of the directly generated CTS $\mathscr{S}$ before optimizations.

Definition 7 (Simulation): Given two state machines $\mathscr{M}_1, \mathscr{M}_2$, and a binary relation τ over $\mathbf{R}_{\mathscr{M}_1}$ and $\mathbf{R}_{\mathscr{M}_2}$. τ is a simulation from $\mathscr{M}_1$ to $\mathscr{M}_2$ if:

1) for each $v \in \mathbf{I}_{\mathscr{M}_1}$, there exists $v' \in \mathbf{I}_{\mathscr{M}_2}$ s.t. $\tau(v, v')$;
2) for all $v_1, v_2 \in \mathbf{R}_{\mathscr{M}_1}$ and all $v'_1 \in \mathbf{R}_{\mathscr{M}_2}$ s.t. $(v_1, v_2) \in \mathbf{T}_{\mathscr{M}_1}$ and $\tau(v_1, v'_1)$, there exists $v'_2 \in \mathbf{R}_{\mathscr{M}_2}$ s.t. $(v'_1, v'_2) \in \mathbf{T}_{\mathscr{M}_2}$ and $\tau(v_2, v'_2)$.

Fig. 6 shows a simulation τ from $\mathscr{M}_1$ to $\mathscr{M}_2$, corresponding to two conditions of simulations.

We declare a binary relation $\approx$ over states of $\mathscr{M}_\mathcal{S}$ and $\mathscr{M}_\mathscr{S}$, and show $\approx$ is a simulation from $\mathscr{M}_\mathcal{S}$ to $\mathscr{M}_\mathscr{S}$, and vice versa.

Definition 8 ($\approx$): Let v, u be any states in $\mathbf{U}_{\mathscr{M}_\mathcal{S}}$ and $\mathbf{U}_{\mathscr{M}_\mathscr{S}}$ respectively. v equals u w.r.t. $\approx$ (denoted by $v \approx u$) if:

- for each o ($\hat{o}$) in $\mathcal{O}$, and each d of D_o (each p of D_p) s.t. $o(v) = d$ ($\hat{o}(v, p) = d$), there exists $(o' \colon\_ : D_{o'} \to D_{oc})$

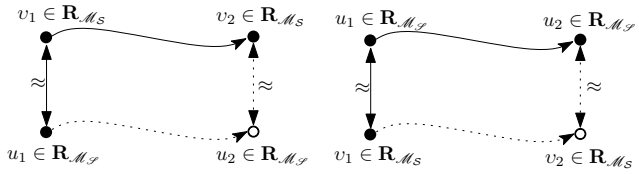


Fig. 7. Diagrams of Lemma 3 and Lemma 4

in C_o $(\hat{o}'[\_]:\_ : D_p\ D_{\hat{o}'} \to D_{oc}$ in $\mathsf{C}_o)$ s.t. $h(o) = (o':\_ : D_{o'} \to D_{oc})$ and $(o' : d)$ is in u $(h(\hat{o}) = \hat{o}'[\_]:\_ : D_p\ D_{\hat{o}'} \to D_{oc}$ and $(\hat{o}'[p] : d)$ is in $u)$;

- for each $(o:\_ : D_o \to D_{oc})$ in C_o $(\hat{o}[\_]:\_ : D_p : D_{\hat{o}} \to D_{oc}$ in $\mathsf{C}_o)$ and $(o : d)$ $(o[p] : d)$ is in u, there exists $(o' : \Upsilon \to D_{o'})$ in $\mathcal{O}$ $(\hat{o}' : \Upsilon\ D_p \to D_{\hat{o}'}$ in $\mathcal{O})$ s.t. $h^{-1}(o:\_ : D_o \to D_{oc}) = (o' : \Upsilon \to D_{o'})$ and $o(v) = d$ $(h^{-1}(\hat{o}:\_ : D_{\hat{o}}\ D_p \to D_{oc}) = (\hat{o}' : \Upsilon\ D_p \to D_{\hat{o}'})$ and $\hat{o}'(v, p) = d)$.

h^{-1} is the inverse function of h. It is obvious that $\approx$ is symmetric. Definition 8 says that there exist one-one correspondences between observed values in v and observational components in u if $v \approx u$.

Lemma 3: for all $v_1, v_2 \in \mathbf{R}_{\mathscr{M}_S}$, and for all $u_1 \in \mathbf{R}_{\mathscr{M}_\mathscr{S}}$ s.t. $(v_1, v_2) \in \mathbf{T}_{\mathscr{M}_S}$ and $v_1 \approx u_1$, there exists $u_2 \in \mathbf{R}_{\mathscr{M}_\mathscr{S}}$ s.t. $(u_1, u_2) \in \mathbf{T}_{\mathscr{M}_\mathscr{S}}$ and $v_2 \approx u_2$.

Lemma 4: for all $u_1, u_2 \in \mathbf{R}_{\mathscr{M}_\mathscr{S}}$, and for all $v_1 \in \mathbf{R}_{\mathscr{M}_S}$ s.t. $(u_1, u_2) \in \mathbf{T}_{\mathscr{M}_\mathscr{S}}$ and $u_1 \approx v_1$, there exists $v_2 \in \mathbf{R}_{\mathscr{M}_S}$ s.t. $(v_1, v_2) \in \mathbf{T}_{\mathscr{M}_S}$ and $u_2 \approx v_2$.

We omit the proofs due to the limitation of space. Interested readers can refer to [10] for the details.

Theorem 2: $\mathscr{M}_\mathscr{S}$ simulates $\mathscr{M}_\mathcal{S}$ w.r.t $\approx$, and vice versa.

Proof: We first show that $\mathscr{M}_\mathscr{S}$ simulates $\mathscr{M}_\mathcal{S}$ w.r.t. $\approx$. It is straightforward to show by the definition of ψ that for each $u \in \mathbf{I}_{\mathscr{M}_\mathscr{S}}$, there exists $v \in \mathbf{I}_{\mathscr{M}_S}$ s.t. $u \approx v$. Lemma 3 shows that $\approx$ satisfies the second condition of simulation. It is proved likewise that $\mathscr{M}_\mathcal{S}$ simulates $\mathscr{M}_\mathscr{S}$ w.r.t. $\approx$. ■

Corollary 1: $\mathscr{M}_\mathcal{S}$ simulates $\mathscr{M}_{\mathscr{S}'}$ w.r.t $\approx$, and vice versa.

C. Proof of invariant-preservation property

Let $\mathscr{B}[\Box_1, \ldots, \Box_n]$ be a general Boolean context, and ρ be a state predicate $\rho : \mathbf{U}_{\mathscr{M}_\mathcal{S}} \to \mathcal{B}$ in $\mathscr{M}_\mathcal{S}$ s.t. $\forall v \in \mathbf{U}_{\mathscr{M}_\mathcal{S}}.\rho(v) \triangleq \mathscr{B}[o_1(v), \ldots, o_{n_1}(v), o_{n_1+1}(v, p_1), \ldots, o_{n_1+n_2}(v, p_{n_2})]$, where $n_1 + n_2 = n$, and $p_i \in D_p$ for $i = 1, \ldots, n_2$.

Let h be the function that takes a state predicate ρ in $\mathscr{M}_\mathcal{S}$ and returns a predicate $h(\rho) : \mathbf{U}_{\mathscr{M}_{\mathscr{S}'}} \to \mathcal{B}$ in $\mathscr{M}_{\mathscr{S}'}$, such that for each u in $\mathbf{R}_{\mathscr{M}_{\mathscr{S}'}}$:

$$h(\rho)(u) \triangleq \begin{cases} \mathscr{B}[z_1, \ldots, z_{n_1}, z_{n_1+1}, \ldots, z_{n_1+n_2}] & \text{if } cond \\ false & \text{otherwise} \end{cases}$$

where, $cond$ means that there exists a substitution δ and a context $\mathscr{C}$ s.t. $u = \mathscr{C}[\delta((o_1 : z_1) \ldots (o_{n_1} : z_{n_1})(o_{n_1+1}[p_1] : z_{n_1+1}) \ldots (o_{n_1+n_2}[p_{n_2}] : z_{n_1+n_2}))]$.

Lemma 5: Given a state predicate ρ, for all $v \in \mathbf{U}_{\mathscr{M}_\mathcal{S}}$ and $u \in \mathbf{U}_{\mathscr{M}_{\mathscr{S}'}}$, $v \approx u$ implies $\rho(v) = h(\rho)(u)$.

Lemma 5 says that $\approx$ preserves state predicate in that $\rho(v)$ is true iff $h(\rho)(u)$ when $v \approx u$. The proof is straightforward because p and $h(\rho)$ only depend upon the same values among them and $v \approx u$ implies they contain the same values.

Corollary 2: A state predicate ρ is an invariant w.r.t. $\mathscr{M}_\mathcal{S}$ iff $h(\rho)$ is an invariant w.r.t. $\mathscr{M}_{\mathscr{S}'}$.

Corollary 2 says that the transformation (including the optimization) preserves invariants. Specifically, when a reachable state u is found in CTS $\mathscr{S}'$ s.t. $h(\rho)(u)$ is false, there is a reachable state v where $\rho(v)$ is false.

V. Related work

Nakamura et al. have proposed a transformation approach from equational theories into rewrite theories for the same purpose as ours. They proved their approach is *complete*, but not *sound* [11]. By completeness they mean that if an invariant holds in original equational theory it holds in generated rewrite theory, and by soundness that the inverse holds. In that sense, the invariant-preservation property of our transformation approach can be considered both *complete* and *sound*.

VI. Conclusion and future work

We have presented an approach to the transformation from a subclass of equational theories to rewrite theories, and proved that the transformation preserves invariant properties. Invariant-preservation means that an invariant holds in an original equational theory $\mathcal{S}$ iff it holds in the generated rewrite theory $\mathscr{S}$. It guarantees that if we can find counterexamples by model checking generated rewrite theories for an invariant property, we do not need to try to prove it in vain by theorem proving. Thus, invariant-preservation is fundamental to the application of the transformation to combined verification by using both theorem proving and model checking [6], [12].

References

[1] J. A. Goguen and K. Lin, "Specifying, programming and verifying with equational logic," in *We Will Show Them! (2)*. College Publications, 2005, pp. 1–38.

[2] J. Meseguer, "Rewriting logic as a semantic framework for concurrency: a progress report," in *7th CONCUR, LNCS 1119*. Springer, 1996, pp. 331–372.

[3] G. Huet and J. Hullot, "Proofs by induction in equational theories with constructors," *Journal of Computer and System Sciences*, vol. 25, no. 2, pp. 239–266, 1982.

[4] M. Clavel, F. Durán, and et al., "All about Maude," *LNCS 4350, Springer*, 2007.

[5] K. Bae and J. Meseguer, "State/event-based LTL model checking under parametric generalized fairness," in *23rd CAV, LNCS 6860*. Springer, 2011, pp. 132–148.

[6] M. Zhang, K. Ogata, and M. Nakamura, "Translation of State Machines from Equational Theories into Rewrite Theories with Tool Support," *IEICE Transactions*, vol. 94-D, no. 5, pp. 976–988, 2011.

[7] R. Diaconescu and K. Futatsugi, *CafeOBJ Report*. World Scientific, 1998.

[8] K. Ogata and K. Futatsugi, "Proof Scores in the OTS/CafeOBJ Method," in *FMOODS'03, LNCS 2884*. Springer, 2003, pp. 170–184.

[9] J. Meseguer, "Conditional Rewriting Logic as a Unified Model of Concurrency," *TCS*, vol. 96, no. 1, pp. 73–155, 1992.

[10] M. Zhang, "A study on generating efficient rewrite theory specification from behavioral specification," 2011, PhD thesis.

[11] M. Nakamura, W. Kong, and et al, "A Specification Translation from Behavioral Specifications to Rewrite Specifications," *IEICE Transactions*, vol. 91-D, no. 5, pp. 1492–1503, 2008.

[12] K. Ogata and K. Futatsugi, "A Combination of Forward & Backward Reachability Analysis Methods," in *12th ICFEM, LNCS 6447*. Springer, 2010, pp. 501–517.