Equivalence checking for NuSMV specifications

Paolo Arcaini[1], Angelo Gargantini[2], and Elvinia Riccobene[1]

[1] Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Italy
{paolo.arcaini,elvinia.riccobene}@unimi.it
[2] Dipartimento di Ingegneria dell'Informazione e Metodi Matematici, Università degli Studi di Bergamo, Italy
angelo.gargantini@unibg.it

Abstract. We present a technique for checking the equivalence of NuSMV specifications. The approach is founded on the notion of equivalence between Kripke structures. The necessity to tackle this problem arisen working on using mutation to asses the static analysis fault detection capability. Indeed, *mutation*, consisting into introducing simple syntactic changes – representing typical mistakes designers often make – into specifications, may produce equivalent *mutants*, namely models behaving as the original one. Equivalent mutants should be detected since they do not represent actual faults. In program mutation, detecting equivalent mutants is an undecidable problem and, when possible, is a time-consuming activity, difficult to automatize. In this work we focus on how detecting equivalence of NuSMV specifications. The novel technique we propose, consists in building a merging unique specification and proving by model checking a series of CTL properties.

1 Introduction

The problem of detecting equivalent NuSMV specifications is connected to the problem of identifying equivalent mutants. *Mutation* consists in introducing small modifications, called *mutations*, into models; these simple syntactic changes should represent typical mistakes a designer may make during the modeling activity.

Mutation and the problem of checking equivalent mutants is well-known in the context of program, and common fault classes have been defined by Kuhn in [10]. From fault classes it is easy to derive *mutation operators* [1], that produce copies of the original program each containing a single fault: these faulty programs are called *mutants*. More recently, mutation has been applied to specifications like FSMs [6], Petri nets [7], Statecharts [8], Estelle specifications [5], Object-Z specifications [11], etc. We have focused our attention on the mutation of Kripke structures and, in particular, of their representation as NuSMV specifications [13].

Most mutation operators can produce equivalent mutants, namely models behaving as the original one. Equivalent mutants pose a challenge, since they do not represent actual faults and cannot be detected by observing the model behavior. Therefore, it is important to identify and remove from the set of mutants the equivalent ones.

In program mutation, detecting equivalent mutants is an undecidable problem [1] and, when possible, is a time-consuming activity [9], difficult to automatize. In this work we focus on how detecting equivalence of NuSMV specifications obtained by mutation. We propose a novel procedure for checking the equivalence, which is based on the notion of equivalence between Kripke structures.

Section 2.1 presents the NuSMV syntax and Section 2.2 introduces some definitions about Kripke structures. Section 3 presents the notion of equivalence between Kripke structures and how the problem of proving the equivalence between two Kripke structures M_1 and M_2 can be reduced to the problem of proving some properties over a single Kripke structure M_{12}, obtained by *merging* M_1 and M_2; Section 4 shows how to build the merging specification when the Kripke structures are represented as NuSMV specifications and how to prove the equivalence using some CTL properties.

2 Background

2.1 NuSMV and its notation

NuSMV [4,13] is known as a model checker derived from the CMU SMV [12]. It allows for the representation of synchronous and asynchronous finite state systems, and for the analysis of specifications expressed in *Computation Tree Logic* (CTL) and *Linear Temporal Logic* (LTL).

A NuSMV specification contains a **VAR** section for variable declarations. A variable type can be Boolean, integer defined over intervals or sets, or an enumeration of symbolic constants. A *state* of the model is an assignment of values to variables.

A NuSMV specification describes the behavior of a Finite State Machine (FSM) in terms of a "possible next state" relation between states that are determined by the values of variables. Transitions between states are determined by the updates of the variables declared in the **ASSIGN** section, that contains the initialization (by the instruction **init**) and the update mechanism (by the instruction **next**) of variables. A **DEFINE** statement can also be used as a macro to syntactically replace an *identifier* with the *expression* it is associated with. There exist the following four ways to explicitly assign values to a variable:

ASSIGN identifier := simple_expression −− *simple assignment*
ASSIGN **init**(identifier) := simple_expression −− *init value*
ASSIGN **next**(identifier) := next_expression −− *next value*
DEFINE identifier := simple_expression −− *macro definition*

where *identifier* is a variable identifier; *simple_expression*s are built only from the values of variables in the current state and they cannot have a **next** operation inside; *next_expression* relates current and next state variables to express transitions in the FSM (see the NuSMV User Manual [3] for more details on the assignment syntax and restriction rules for assignments). In both *simple*- and *next*- expressions, a variable's value can be determined either unconditionally or conditionally, depending on the form of the expression. Conditional expressions can be:

1. An **if-then-else** expression cond1? exp1: exp2 which evaluates to *exp1* if the condition *cond1* evaluates to true, and to *exp2* otherwise.

2. A condition **case** expression:

case
left_expression_1 : right_expression_1 ;
...
left_expression_N : rightexpression_N;
esac

which returns the value of the first *right_expression_i* such that the corresponding *left_expression_i* condition evaluates to TRUE, and the previous *i-1* left expressions evaluate to FALSE. The type of expressions on the left hand side must be boolean. An error occurs if all expressions on the left hand side evaluate to FALSE. To avoid these kinds of errors, NuSMV performs a static analysis and, if it believes that in some states no left expression may be true, it forces the user to add a *default case* with *left_expression* equal to TRUE.

NuSMV offers another more declarative way of defining initial states and transition relations. Initial states can be defined by the keyword **INIT** followed by characteristic properties that must be satisfied by the variables values in the initial states. Transition relations can be expressed by constraints, through the keyword **TRANS**, on a set of *current state/next state* pairs. *Invariant conditions* can be expressed by the command **INVAR**.

Temporal properties are specified in the **CTLSPEC** (resp. **LTLSPEC**) section that contains the CTL (resp. LTL) properties to be verified.

2.2 Kripke structures

Definition 1 (Kripke structure).
*A **Kripke structure** is a quadruple* $M = \langle S, S^0, T, \mathcal{L} \rangle$ *where*

- S *is a set of states;*
- $(S^0 \subseteq S) \neq \emptyset$ *is the set of initial states;*
- $T \subseteq S \times S$ *is the transition relation that must be left-total, i.e.,* $\forall s \in S, \exists s' \in S \colon (s, s') \in T$*;*
- $\mathcal{L} : S \to \mathcal{P}(AP)$ *is the proposition labeling function, where* AP *is a set of atomic propositions; we require* $\mathcal{L}$ *to be injective, i.e.,* $\forall s_1, s_2 \in S,\ s_1 \neq s_2 \to \mathcal{L}(s_1) \neq \mathcal{L}(s_2)$*: this means that a state is uniquely identified by its labels.*

Definition 2 (Computation tree). *Given a Kripke structure* $M = (S, S^0, T, \mathcal{L})$*, a computation tree of* M *is a tree structure where the root is an initial state* $s_0 \in S_0$*, and the children of a node* $s \in S$ *in the computation tree are all the states* $s' \in S$ *such that there exists a transition* $(s, s') \in T$*.*

Definition 3 (Structure equivalence).
Two Kripke structures M_1 *and* M_2 *with the same set of atomic propositions are equivalent iff they have the same computation trees.*

Definition 4 (Path).
A path ψ *is a sequence of states in* S

$$\psi = s_1, s_2, \ldots, s_n$$

such that

$$\forall i \in [1, n-1] \quad (s_i, s_{i+1}) \in T$$

Let's identify with Ψ *the (infinite) set of all the paths in* M*.*
Let's identify with $\Psi^0 \subseteq \Psi$ *the (infinite) set of all the paths such that the starting state* $s_1 \in S^0$*.*

Definition 5 (Reachability).
A state $s \in S$ *is reachable in* M *if there exists a path* $\psi^0 = s_1, \ldots, s_n \in \Psi^0$ *such that* $s_n = s$*, i.e.*

$$isReach(s) \triangleq \exists \psi^0 = s_1, \ldots, s_n \in \Psi^0 \colon s_n = s$$

We denote by $reach(M) \subseteq S$ *the set of reachable states of the machine* M*.*

Definition 6 (Successor state).
A state s' *is a* successor *of another state* s *if* $(s, s') \in T$*. We denote by* $next(s)$ *the set of the successor states of* s*, i.e.*

$$next(s) = \{s' \in S \colon (s, s') \in T\}$$

3 Equivalence of Kripke structures

In this section we give the notion of equivalence between two Kripke structures [2].

Let $M_1 = \langle S_1, S_1^0, T_1, \mathcal{L}_1 \rangle$ and $M_2 = \langle S_2, S_2^0, T_2, \mathcal{L}_2 \rangle$ be two Kripke structures with the same set of atomic propositions AP. A relation E can be defined on $S_1 \times S_2$ to express the equivalence between states of the two structures M_1 and M_2; two states are equivalent if they have the same labels and bring to next states having the same labels.

Definition 7 (State equivalence).
$\forall s_1 \in S_1 \forall s_2 \in S_2$ we say $s_1 E s_2$ iff the following condition holds:

$$\begin{array}{ll} \mathcal{L}_1(s_1) = \mathcal{L}_2(s_2) & \wedge \\ \forall s'_1 \in next(s_1)\ \exists s'_2 \in next(s_2) : \mathcal{L}_1(s'_1) = \mathcal{L}_2(s'_2) & \wedge \\ \forall s'_2 \in next(s_2)\ \exists s'_1 \in next(s_1) : \mathcal{L}_2(s'_2) = \mathcal{L}_1(s'_1) & \end{array} \tag{1}$$

Theorem 1 (Structure equivalence).
Let M_1 and M_2 be two Kripke structures with the same set of atomic propositions. If the following properties hold (initial states have same labeling and reachable states are equivalent):

$$\forall s_1^0 \in S_1^0, \exists s_2^0 \in S_2^0 \colon \left[\mathcal{L}_1(s_1^0) = \mathcal{L}_2(s_2^0)\right] \tag{2}$$

$$\forall s_2^0 \in S_2^0, \exists s_1^0 \in S_1^0 \colon \left[\mathcal{L}_2(s_2^0) = \mathcal{L}_1(s_1^0)\right] \tag{3}$$

$$\forall s_1 \in reach(M_1)\ \exists s_2 \in reach(M_2) \colon s_1\ E\ s_2 \tag{4}$$

$$\forall s_2 \in reach(M_2)\ \exists s_1 \in reach(M_1) \colon s_2\ E\ s_1 \tag{5}$$

then M_1 and M_2 are equivalent.

The problem of checking the equivalence of two Kripke structures M_1, M_2 can be reduced to the problem of proving some properties over a new *merging* Kripke structure M_{12} derived from M_1 and M_2. In Section 3.1 we show how to build M_{12}.

3.1 Construction of merging Kripke structure M_{12}

Let $M_1 = \langle S_1, S_1^0, T_1, \mathcal{L}_1 \rangle$ and $M_2 = \langle S_2, S_2^0, T_2, \mathcal{L}_2 \rangle$ be two Kripke structures with the same set of atomic propositions AP.

Let $M_{12} = \langle S_{12}, S_{12}^0, T_{12}, \mathcal{L}_{12} \rangle$ be a Kripke structure built upon M_1 and M_2, satisfying the following conditions.

***C1**: condition over the states S_{12}.* There exist two *projection* functions:

$$\pi_1 \colon S_{12} \to S_1 \tag{6}$$

$$\pi_2 \colon S_{12} \to S_2 \tag{7}$$

such that

$$\forall s_1 \in S_1, \forall s_2 \in S_2, \exists s_{12} \in S_{12}\ \left[\pi_1(s_{12}) = s_1 \wedge \pi_2(s_{12}) = s_2\right] \tag{8}$$

***C2**: condition over the initial states S_{12}^0.*

$$\forall s \in S_{12}\ \left[s \in S_{12}^0 \iff \pi_1(s) \in S_1^0 \wedge \pi_2(s) \in S_2^0\right] \tag{9}$$

***C3**: condition over the transition relation T_{12}.*

$$\forall s_{12} \in S_{12}, \forall s'_{12} \in S_{12}\ \left[s'_{12} \in next_{M_{12}}(s_{12}) \iff \begin{array}{l} \pi_1(s'_{12}) \in next_{M_1}(\pi_1(s_{12})) \wedge \\ \pi_2(s'_{12}) \in next_{M_2}(\pi_2(s_{12})) \end{array} \right] \tag{10}$$

3.1.1 Corollary

A state $s_{12} \in S_{12}$ is reachable in M_{12} iff its projections $\pi_1(s_{12})$ and $\pi_2(s_{12})$ are, respectively, reachable in M_1 and M_2.

$$isReach_{M_{12}}(s_{12}) \iff isReach_{M_1}(\pi_1(s_{12})) \wedge isReach_{M_2}(\pi_2(s_{12})) \tag{11}$$

Proof Let's suppose that

$$\exists s_{12} \in S_{12}\ [isReach_{M_{12}}(s_{12}) \wedge \neg isReach_{M_2}(\pi_2(s_{12}))] \tag{12}$$

If s_{12} is reachable in M_{12}, it means that there exists a path $\psi_{12}^0 = s_1, \ldots, s_n \in \Psi_{12}^0$ such that $s_n = s_{12}$.

Let's now consider the following sequence of states in M_2:

$$\pi_2(s_1), \ldots, \pi_2(s_n)$$

By Formula 12 we know that the projection of state s_n in M_2 is not reachable, i.e., $\neg isReach_{M_2}(\pi_2(s_n))$[3]; this means that $\exists i \in [1, n-1]$: $\pi_2(s_{i+1}) \notin next(\pi_2(s_i))$. But this contradicts the condition on the construction of T_{12} (formula 10). A similar contradiction is achieved if, at the beginning of the proof, we suppose that

$$\exists s_{12} \in S_{12}\ [isReach_{M_{12}}(s_{12}) \wedge \neg isReach_{M_1}(\pi_1(s_{12}))]$$

3.2 Equivalence checking of Kripke structures

Definition 8 (Equivalence of the projections).

We say that a state $s_{12} \in S_{12}$ is labelly equivalent *iff the labelings of the projections are equivalent, i.e*

$$le(s_{12}) \triangleq \mathcal{L}_1(\pi_1(s_{12})) = \mathcal{L}_2(\pi_2(s_{12})) \tag{13}$$

We say that two states $s_{12}, s'_{12} \in S_{12}$ are labelly equivalent *with respect to the projection π_1/π_2 iff the labelings of the projections are equivalent, i.e*

$$le_1(s_{12}, s'_{12}) \triangleq \mathcal{L}_1(\pi_1(s_{12})) = \mathcal{L}_1(\pi_1(s'_{12})) \tag{14}$$

$$le_2(s_{12}, s'_{12}) \triangleq \mathcal{L}_2(\pi_2(s_{12})) = \mathcal{L}_2(\pi_2(s'_{12})) \tag{15}$$

Definition 9 (Mirror state).

For all states $s_{12} \in S_{12}$ we define the predicate mirror *as:*

$$mirror(s_{12}) \ \triangleq\ le(s_{12}) \rightarrow \forall s'_{12} \in next(s_{12}) \begin{pmatrix} \exists s''_{12} \in next(s_{12})\ [le(s''_{12}) \wedge le_1(s'_{12}, s''_{12})] \wedge \\ \exists s'''_{12} \in next(s_{12})\ [le(s'''_{12}) \wedge le_2(s'_{12}, s'''_{12})] \end{pmatrix} \tag{16}$$

Theorem 2 (Equivalence between M_1 and M_2).

M_1 and M_2 are equivalent iff the following properties

$$\forall s_{12}^0 \in S_{12}^0,\ \exists {s_{12}^0}' \in S_{12}^0\ [le({s_{12}^0}') \wedge le_1(s_{12}^0, {s_{12}^0}')] \tag{17}$$

$$\forall s_{12}^0 \in S_{12}^0,\ \exists {s_{12}^0}'' \in S_{12}^0\ [le({s_{12}^0}'') \wedge le_2(s_{12}^0, {s_{12}^0}'')] \tag{18}$$

$$\forall s_{12} \in reach(M_{12})\ [mirror(s_{12})] \tag{19}$$

hold in M_{12}.

[3] $s_{12} = s_n$

4 Equivalence checking of NuSMV specifications

Definition 10 (NuSMV model as Kripke structure).
A NuSMV model is a Kripke stucture $M = \langle S, S^0, T\rangle$ where each state of S is labeled by a predicate $\bigwedge_{i=1}^{r}(v_i = d_i)$, being $var(M) = \{v_1, \ldots, v_r\}$ a finite fixed set of variables and $\{d_1, \ldots, d_r\}$ their interpretation values over domains $D_1, \ldots, D_r$; the transition relation T expresses the updating of the state variables interpretation by the syntax given in Section 2.1.

4.1 Mutated specification

Given a NuSMV model $M_o = \langle S_o, S_o^0, T_o\rangle$, we apply a *mutation* to it: we mutate the initial assignments and/or the next state assignments of a set of variables, that is the way in which their initial/next value is calculated. We obtain a machine $M_{mu} = \langle S_{mu}, S_{mu}^0, T_{mu}\rangle$ with the same state space and the same variables, i.e., $S_o = S_{mu}$ and $var(M_o) = var(M_{mu})$, but, maybe, with a different transition relation T_{mu} and/or a different set of initial states S_{mu}^0. If $S_o^0 = S_{mu}^0 \wedge T_o^0 = T_{mu}^0$, the two models M_o and M_{mu} are said to be *equivalent*, otherwise they are *not equivalent*.

Partitioning of the variables The variables $var(M_o)$ can be decomposed in subsets depending on the fact that they are *affected* by the mutation or not.

Let $MV = \{\tilde{v}_1, \ldots, \tilde{v}_k\}$ be the set of variables whose initial/next assignment has been mutated. Let $\tilde{D}_1, \ldots, \tilde{D}_k$ be their domains.

Let NV be the set of variables whose initial/next assignment has not been mutated. NV can be decomposed in two parts:

- the set of variables $DV = \{v_{k+1}, \ldots, v_n\}$: a variable $v_j \in DV$ iff there is a variable $v_i \in MV$ whose value, in some state, is determined according to the value of v_j in the current/previous state. Let $D_{k+1}, \ldots, D_n$ be their domains;
- the set of variables IN that are not considered in the evaluation of the value of any variable in MV.

The variables of M_o and of M_{mu} are the same, i.e.:

$$var(M_o) = var(M_{mu}) = MV \cup NV = MV \cup DV \cup IN$$

Example Model 1 shows a NuSMV specification and Model 2 another specification obtained from the previous one applying a mutation to it: the relation operator $\geqslant$ in the second branch of the case expression in the assignment of variable *amPm* has been replaced with the relational operator $>$. The obtained partition of the variables is:

- MV= {amPm}
- DV= {h}
- IN= {h12}

Model 3 shows the specification obtained from the merging of the original specification (Model 1) and the mutated one (Model 2). Assignments of variables MV ($amPm$) and DV (h) are those defined in the original specification. Variables MV' ($amPmMut$) have been obtained introducing a copy of variables in MV, appending the suffix *Mut* to their names; their assignments are those defined in the mutated specification. Variables in IN ($h12$) have not been exported.

Model 1	Model 2	Model 3
MODULE main **VAR** h: 0..23; h12: 1..12; amPm: {AM, PM}; **ASSIGN** **init**(h) := 0; **next**(h) := (h + 1) **mod** 24; h12 := **case** h **in** {1, 12}: h; h > 12: h **mod** 12; **TRUE**: 12; **esac**; amPm := **case** h < 12: AM; h >= 11: PM; **esac**;	**MODULE** main **VAR** h: 0..23; h12: 1..12; amPm: {AM, PM}; **ASSIGN** **init**(h) := 0; **next**(h) := (h + 1) **mod** 24; h12 := **case** h **in** {1, 12}: h; h > 12: h **mod** 12; **TRUE**: 12; **esac**; amPm := **case** h < 12: AM; h > 11: PM; **esac**;	**MODULE** main **VAR** h: 0..23; amPm: {AM, PM}; amPmMut: {AM, PM}; **ASSIGN** **init**(h) := 0; **next**(h) := (h + 1) **mod** 24; amPm := **case** h < 12: AM; h >= 11: PM; **esac**; amPmMut := **case** h < 12: AM; h > 11: PM; **esac**;
Model 1. Original specification	**Model 2.** Equivalent mutant	**Model 3.** Merging specification

4.2 Merging specification

Given the NuSMV specifications M_o and M_{mu}, we define **merging** specification the NuSMV model $M_e = \langle S_e, S_e^0, T_e \rangle$ built as follows:

- $var(M_e) = MV \cup MV' \cup DV$, being
  - $MV = \{\tilde{v}_1, \ldots, \tilde{v}_k\}$ the set of all variables of M_o whose initial assignment and/or next state assignment has been mutated in M_{mu}. $\tilde{D}_1, \ldots, \tilde{D}_k$ are their domains.
  - $MV' = \{\tilde{v}'_1, \ldots, \tilde{v}'_k\}$ a renamed copy of MV. Their domains are the same of the variables in MV, that is $\tilde{D}_1, \ldots, \tilde{D}_k$. There exists a bijective function

    $$mut : MV \to MV'$$

    such that $\forall \tilde{v}_i \in MV(mut(\tilde{v}_i) = \tilde{v}'_i)$.
  - $DV = \{v_{k+1}, \ldots, v_n\}$ the set of all non mutated variables of M_o upon which the value of some mutated variable depends on.
- The initial state assignments of variables in $MV \cup DV$ are those defined in M_o, while variables in MV' have initial assignments as in M_{mu}.
- The next state assignments of variables in $MV \cup DV$ are those defined in M_o, while variables in MV' have next state assignments as in M_{mu}.

4.3 Equivalence of the projections and mirror state

Since a state in a NuSMV model is identified by the values of its variables, for NuSMV specifications, the predicates le, le_1 and le_2 (see Definition 8) can be defined in the following way:

$$\begin{aligned} le(s_e) &\triangleq \forall v \in MV \ \ [[\![v]\!]_{s_e} = [\![mut(v)]\!]_{s_e}] \\ &\triangleq \bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s_e} \end{aligned} \tag{20}$$

$$le_1(s_e, s'_e) \triangleq \forall v \in (MV \cup DV)\ \left[[\![v]\!]_{s_e} = [\![v]\!]_{s'_e} \right] \triangleq \bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}_i]\!]_{s'_e} \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s_e} = [\![v_j]\!]_{s'_e} \tag{21}$$

$$le_2(s_e, s'_e) \triangleq \forall v \in (MV' \cup DV)\ \left[[\![v]\!]_{s_e} = [\![v]\!]_{s'_e} \right] \triangleq \bigwedge_{i=1}^{k} [\![\tilde{v}'_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s'_e} \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s_e} = [\![v_j]\!]_{s'_e} \tag{22}$$

Applying formulas 20 and 21, the formula $le(s'_e) \wedge le_1(s_e, s'_e)$ can be written in the following way:

$$\begin{aligned} le(s'_e) \wedge le_1(s_e, s'_e) &\triangleq \bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s'_e} = [\![\tilde{v}'_i]\!]_{s'_e} \wedge \bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}_i]\!]_{s'_e} \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s_e} = [\![v_j]\!]_{s'_e} \\ &\triangleq \bigwedge_{i=1}^{k} \left([\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}_i]\!]_{s'_e} \wedge [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s'_e} \right) \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s_e} = [\![v_j]\!]_{s'_e} \end{aligned} \tag{23}$$

Applying formulas 20 and 22, the formula $le(s'_e) \wedge le_2(s_e, s'_e)$ can be written in the following way:

$$\begin{aligned} le(s'_e) \wedge le_2(s_e, s'_e) &\triangleq \bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s'_e} = [\![\tilde{v}'_i]\!]_{s'_e} \wedge \bigwedge_{i=1}^{k} [\![\tilde{v}'_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s'_e} \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s_e} = [\![v_j]\!]_{s'_e} \\ &\triangleq \bigwedge_{i=1}^{k} \left([\![\tilde{v}'_i]\!]_{s_e} = [\![\tilde{v}_i]\!]_{s'_e} \wedge [\![\tilde{v}'_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s'_e} \right) \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s_e} = [\![v_j]\!]_{s'_e} \end{aligned} \tag{24}$$

Finally, the predicate *mirror* (see Definition 9) for NuSMV specifications can be defined using formulas 20, 23 and 24.

$$\begin{aligned} & mirror(s_e) \triangleq \\ & \left(\bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s_e} \right) \rightarrow \\ & \forall s'_e \in next(s_e) \\ & \left(\begin{array}{l} \exists s''_e \in next(s_e) \left[\bigwedge_{i=1}^{k} \left([\![\tilde{v}_i]\!]_{s'_e} = [\![\tilde{v}_i]\!]_{s''_e} \wedge [\![\tilde{v}_i]\!]_{s'_e} = [\![\tilde{v}'_i]\!]_{s''_e} \right) \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s'_e} = [\![v_j]\!]_{s''_e} \right] \wedge \\ \exists s'''_e \in next(s_e) \left[\bigwedge_{i=1}^{k} \left([\![\tilde{v}'_i]\!]_{s'_e} = [\![\tilde{v}_i]\!]_{s'''_e} \wedge [\![\tilde{v}'_i]\!]_{s'_e} = [\![\tilde{v}'_i]\!]_{s'''_e} \right) \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s'_e} = [\![v_j]\!]_{s'''_e} \right] \end{array} \right) \end{aligned} \tag{25}$$

4.4 Equivalence checking through CTL properties

Definition 11 (*Both* and *Either* predicates).

Let

$$\begin{aligned} Both(\tilde{d}^k_{i=1}, d^n_{j=k+1}) &\triangleq \bigwedge_{i=1}^{k} \left(\tilde{d}_i = \tilde{v}_i \wedge \tilde{d}_i = \tilde{v}'_i \right) \wedge \bigwedge_{j=k+1}^{n} d_j = v_j \\ Either(\tilde{d}^k_{i=1}, d^n_{j=k+1}) &\triangleq \left(\bigwedge_{i=1}^{k} \tilde{d}_i = \tilde{v}_i \vee \bigwedge_{i=1}^{k} \tilde{d}_i = \tilde{v}'_i \right) \wedge \bigwedge_{j=k+1}^{n} d_j = v_j \end{aligned}$$

be two predicates such that, given a n-upla of values $d = (\tilde{d}^k_{i=1}, d^n_{j=k+1})$*, Both*$(d)$ *means that both machines* M_o *and* M_{mu} *are in the same state* d*, while Either*(d) *means that at least one machine is in state* d*.*

Let's see now how the formulas described in Theorem 2 can be checked through some CTL properties. Section 4.4.1 describes how to prove properties 17 and 18, Section 4.4.2 how to prove property 19.

4.4.1 Condition on the initial states

Definition 12 (Initial state as tuple of values).

Let IS be the tuples of values of the variables in the initial states, i.e.,

$$IS = \left\{ \begin{array}{l} \left(\tilde{d}^k_{i=1} \in \tilde{D}_i, \tilde{d}'^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right) : \\ \qquad \exists s^o_e \in S^0_e \left[\bigwedge_{i=1}^{k} \left(\tilde{d}_i = [\![\tilde{v}_i]\!]_{s^0_e} \wedge \tilde{d}'_i = [\![\tilde{v}'_i]\!]_{s^0_e}\right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s^0_e} \right] \end{array} \right\}$$

Let's also define

$$IS_{MV} = \left\{ \left(\tilde{d}^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right) : \exists s^o_e \in S^0_e \left[\bigwedge_{i=1}^{k} \tilde{d}_i = [\![\tilde{v}_i]\!]_{s^0_e} \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s^0_e} \right] \right\}$$

$$IS_{MV'} = \left\{ \left(\tilde{d}^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right) : \exists s^o_e \in S^0_e \left[\bigwedge_{i=1}^{k} \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s^0_e} \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s^0_e} \right] \right\}$$

By definition of IS, IS_{MV} and $IS_{MV'}$, it holds that

$$\begin{array}{l} \forall \left(\tilde{d}^k_{i=1} \in \tilde{D}_i, \tilde{d}'^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right) \\ \qquad \left(\tilde{d}^k_{i=1}, \tilde{d}'^k_{i=1}, d^n_{j=k+1}\right) \in IS \iff \left(\left(\tilde{d}^k_{i=1}, d^n_{j=k+1}\right) \in IS_{MV} \wedge \left(\tilde{d}'^k_{i=1}, d^n_{j=k+1}\right) \in IS_{MV'}\right) \end{array}$$

First condition on the initial states Using formula 23, formula 17 becomes

$$\forall s^0_e \in S^0_e,\ \exists s^{0'}_e \in S^0_e \left[\bigwedge_{i=1}^{k} \left([\![\tilde{v}_i]\!]_{s^0_e} = [\![\tilde{v}_i]\!]_{s^{0'}_e} \wedge [\![\tilde{v}_i]\!]_{s^0_e} = [\![\tilde{v}'_i]\!]_{s^{0'}_e}\right) \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s^0_e} = [\![v_j]\!]_{s^{0'}_e} \right] \tag{26}$$

Formula 26 can be rewritten, substituting the quantification over the initial states with the quantification over the values of the variables in the initial states (i.e., IS), in the following way

$$\forall (\tilde{d}^k_{i=1}, \tilde{d}'^k_{i=1}, d^n_{j=k+1}) \in IS,\ \exists s^{0'}_e \in S^0_e \left[\bigwedge_{i=1}^{k} \left(\tilde{d}_i = [\![\tilde{v}_i]\!]_{s^{0'}_e} \wedge \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s^{0'}_e}\right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s^{0'}_e} \right] \tag{27}$$

Note that the interpretations of the variables in state s_e have been replaced with the actual values of the variables in the state.

Formula 27 can be further simplified, observing that the values of the variables in MV' (i.e., $\tilde{d}'^k_{i=1}$) are not used in the propositional formula (matrix) of the existentially quantified subformula. Quantifying over IS_{MV}, formula 27 can be rewritten in the following way

$$\forall (\tilde{d}^k_{i=1}, d^n_{j=k+1}) \in IS_{MV},\ \exists s^{0'}_e \in S^0_e \left[\bigwedge_{i=1}^{k} \left(\tilde{d}_i = [\![\tilde{v}_i]\!]_{s^{0'}_e} \wedge \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s^{0'}_e}\right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s^{0'}_e} \right] \tag{28}$$

Second condition on the initial states Using formula 24, formula 18 becomes

$$\forall s_e^0 \in S_e^0,\ \exists s_e^{0''} \in S_e^0 \left[\bigwedge_{i=1}^{k} \left([\![\tilde{v}'_i]\!]_{s_e^0} = [\![\tilde{v}_i]\!]_{s_e^{0''}} \wedge [\![\tilde{v}'_i]\!]_{s_e^0} = [\![\tilde{v}'_i]\!]_{s_e^{0''}} \right) \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s_e^0} = [\![v_j]\!]_{s_e^{0''}} \right] \tag{29}$$

Formula 29 can be rewritten, substituting the quantification over the initial states with the quantification over the values of the variables in the initial states (i.e., IS), in the following way

$$\forall (\tilde{d}_{i=1}^{k}, \tilde{d}'^{k}_{i=1}, d_{j=k+1}^{n}) \in IS,\ \exists s_e^{0''} \in S_e^0 \left[\bigwedge_{i=1}^{k} \left(\tilde{d}'_i = [\![\tilde{v}_i]\!]_{s_e^{0''}} \wedge \tilde{d}'_i = [\![\tilde{v}'_i]\!]_{s_e^{0''}} \right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s_e^{0''}} \right] \tag{30}$$

Formula 30 can be further simplified, observing that the values of the variables in MV (i.e., $\tilde{d}_{i=1}^{k}$) are not used in the matrix of the existentially quantified subformula. Quantifying over $IS_{MV'}$, formula 30 can be rewritten in the following way

$$\forall (\tilde{d}_{i=1}^{k}, d_{j=k+1}^{n}) \in IS_{MV'},\ \exists s_e^{0''} \in S_e^0 \left[\bigwedge_{i=1}^{k} \left(\tilde{d}_i = [\![\tilde{v}_i]\!]_{s_e^{0''}} \wedge \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s_e^{0''}} \right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s_e^{0''}} \right] \tag{31}$$

Unique formula for checking formulas 28 and 31 The matrices of the universally quantified formulas 28 and 31 are the same. So, it is possible to prove both properties, using the following formula.

$$\forall (\tilde{d}_{i=1}^{k} \in \tilde{D}_i, d_{j=k+1}^{n} \in D_j) \left(\begin{array}{l} \exists s_e^0 \in S_e^0 \left[\left(\bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e^0} = \tilde{d}_i \vee \bigwedge_{i=1}^{k} [\![\tilde{v}'_i]\!]_{s_e^0} = \tilde{d}_i \right) \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s_e^0} = d_j \right] \rightarrow \\ \exists s_e^{0'} \in S_e^0 \left[\bigwedge_{i=1}^{k} \left(\tilde{d}_i = [\![\tilde{v}_i]\!]_{s_e^{0'}} \wedge \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s_e^{0'}} \right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s_e^{0'}} \right] \end{array} \right) \tag{32}$$

The proof of the correctness is based on the following theorem.

Theorem 3. *Being A, B and C three domains such that $A \cup B \subseteq C$, it holds that*

$$\forall x \in A\ \ [f(x)] \wedge \forall y \in B\ \ [f(y)] \ \equiv\ \forall z \in C\ \ [(z \in A \vee z \in B) \rightarrow f(z)]$$

In our case $IS_{MV} \subseteq \left(\times_{i=1}^{k} \tilde{D}_i \times \times_{j=k+1}^{n} D_j \right)$ and $IS_{MV'} \subseteq \left(\times_{i=1}^{k} \tilde{D}_i \times \times_{j=k+1}^{n} D_j \right)$. So, we can take as C the domain $\times_{i=1}^{k} \tilde{D}_i \times \times_{j=k+1}^{n} D_j$.

Formula 32 can be rewritten using the *Both* and *Either* predicates (see Definition 11), in the following way

$$\forall \left(\tilde{d}_{i=1}^{k} \in \tilde{D}_i, d_{j=k+1}^{n} \in D_j \right) \left(\begin{array}{l} \exists s_e^0 \in S_e^0\ [\![Either(\tilde{d}_{i=1}^{k}, d_{j=k+1}^{n})]\!]_{s_e^0} \rightarrow \\ \qquad \exists s_e^{0'} \in S_e^0\ [\![Both(\tilde{d}_{i=1}^{k}, d_{j=k+1}^{n})]\!]_{s_e^{0'}} \end{array} \right) \tag{33}$$

4.4.1.1 ***Checking equivalence of initial states in NuSMV*** In NuSMV, a CTL property φ is true iff it is true starting from each initial state, i.e.,

$$M \models \varphi \qquad \text{iff} \qquad \forall s_0 \in S^0\ (M, s_0) \models \varphi$$

So, if we want to know if a property is true in *at least* an initial state, we must check $\neg\varphi$; if $M \not\models \neg\varphi$, it means that there exists an initial state in which φ is true, i.e.,

$$M \not\models \neg\varphi \qquad \text{iff} \qquad \exists s_0 \in S^0\ (M, s_0) \models \varphi$$

So, in order to check the validity of Property 33, $\forall\left(\tilde{d}^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right)$, we first check the CTL property $\neg Either\left(\tilde{d}^k_{i=1}, d^n_{j=k+1}\right)$; if it is false, then we must also check that the CTL property $\neg Both\left(\tilde{d}^k_{i=1}, d^n_{j=k+1}\right)$ is false.

Example In the following, we report some of the CTL properties that must be checked over the specification shown in Model 3 in order to prove the equivalence in the initial states of the specifications shown in Models 1 and 2.

```
CTLSPEC NAME isNotInitState_1 := !((AM = amPm | AM = amPmMut) & h = 0)
CTLSPEC NAME notEqInitState_1 := !((AM = amPm & AM = amPmMut) & h = 0)
CTLSPEC NAME isNotInitState_2 := !((AM = amPm | AM = amPmMut) & h = 1)
CTLSPEC NAME notEqInitState_2 := !((AM = amPm & AM = amPmMut) & h = 1)
--...
CTLSPEC NAME isNotInitState_24 := !((PM = amPm | PM = amPmMut) & h = 0)
CTLSPEC NAME notEqInitState_24 := !((PM = amPm & PM = amPmMut) & h = 0)
CTLSPEC NAME isNotInitState_25 := !((PM = amPm | PM = amPmMut) & h = 1)
CTLSPEC NAME notEqInitState_25 := !((PM = amPm & PM = amPmMut) & h = 1)
...
```

We must check that, if a CTL property *isNotInitState_i* is false, then also the CTL property *notEqInitState_i* is false. In the example, we checked that *isNotInitState_1* and *notEqInitState_1* are false, and all the properties *isNotInitState_i*, with $i = 2, \ldots, 48$, are true: so the two specifications are equivalent in the initial states. Totally we had to check 49 over 96 properties.

4.4.2 Condition on the transitions

Definition 13 (Next state as tuple of values).
Let $NS(s)$ be the set of tuples of values of the variables in the next states of $s \in S$, i.e.

$$NS(s) = \left\{ \begin{array}{l} \left(\tilde{d}^k_{i=1} \in \tilde{D}_i, \tilde{d}'^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right) : \\ \qquad \exists s' \in next(s) \left[\bigwedge_{i=1}^{k} \left(\tilde{d}_i = [\![\tilde{v}_i]\!]_{s'} \wedge \tilde{d}'_i = [\![\tilde{v}'_i]\!]_{s'}\right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s'}\right] \end{array} \right\}$$

Let's also define

$$NS_{MV}(s) = \left\{ \left(\tilde{d}^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right) : \exists s' \in next(s) \left[\bigwedge_{i=1}^{k} \tilde{d}_i = [\![\tilde{v}_i]\!]_{s'} \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s'}\right] \right\}$$

$$NS_{MV'}(s) = \left\{ \left(\tilde{d}^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right) : \exists s' \in next(s) \left[\bigwedge_{i=1}^{k} \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s'} \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s'} \right] \right\}$$

By definition of NS, NS_{MV} and $NS_{MV'}$, it holds that

$$\begin{aligned} &\forall s \in S,\ \forall \left(\tilde{d}^k_{i=1} \in \tilde{D}_i, \tilde{d}'^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right) \\ &\left(\tilde{d}^k_{i=1}, \tilde{d}'^k_{i=1}, d^n_{j=k+1}\right) \in NS(s) \iff \left(\left(\tilde{d}^k_{i=1}, d^n_{j=k+1}\right) \in NS_{MV}(s) \wedge \left(\tilde{d}'^k_{i=1}, d^n_{j=k+1}\right) \in NS_{MV'}(s)\right) \end{aligned}$$

Condition in the transition relation Applying the definition of the predicate *mirror* (see formula 25), formula 19 becomes

$$\begin{aligned} &\forall s_e \in reach(M_e) \\ &\left(\left(\bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s_e} \right) \rightarrow \right. \\ &\forall s'_e \in next(s_e) \\ &\left. \left(\begin{array}{l} \exists s''_e \in next(s_e) \left[\bigwedge_{i=1}^{k} \left([\![\tilde{v}_i]\!]_{s'_e} = [\![\tilde{v}_i]\!]_{s''_e} \wedge [\![\tilde{v}_i]\!]_{s'_e} = [\![\tilde{v}'_i]\!]_{s''_e} \right) \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s'_e} = [\![v_j]\!]_{s''_e} \right] \wedge \\ \exists s'''_e \in next(s_e) \left[\bigwedge_{i=1}^{k} \left([\![\tilde{v}'_i]\!]_{s'_e} = [\![\tilde{v}_i]\!]_{s'''_e} \wedge [\![\tilde{v}'_i]\!]_{s'_e} = [\![\tilde{v}'_i]\!]_{s'''_e} \right) \wedge \bigwedge_{j=k+1}^{n} [\![v_j]\!]_{s'_e} = [\![v_j]\!]_{s'''_e} \right] \end{array} \right) \right) \end{aligned} \tag{34}$$

Formula 34 can be simplified, replacing the universal quantification over the next states of s_e with the universal quantification over the values of the variables in the next states of s_e (i.e., $NS(s_e)$), in the following way

$$\forall s_e \in reach(M_e) \left(\begin{array}{l} \left(\bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s_e} \right) \rightarrow \\ \forall \left(\tilde{d}^k_{i=1} \in \tilde{D}_i, \tilde{d}'^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right) \in NS(s_e) \\ \left(\begin{array}{l} \exists s''_e \in next(s_e) \left[\bigwedge_{i=1}^{k} \left(\tilde{d}_i = [\![\tilde{v}_i]\!]_{s''_e} \wedge \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s''_e} \right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s''_e} \right] \wedge \\ \exists s'''_e \in next(s_e) \left[\bigwedge_{i=1}^{k} \left(\tilde{d}'_i = [\![\tilde{v}_i]\!]_{s'''_e} \wedge \tilde{d}'_i = [\![\tilde{v}'_i]\!]_{s'''_e} \right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s'''_e} \right] \end{array} \right) \end{array} \right) \tag{35}$$

In formula 35, in the first existentially quantified subformula, the values of the variables MV' (i.e., $\tilde{d}'^k_{i=1}$) are never used, and, in the second existentially quantified subformula, the values of the variables MV (i.e., $\tilde{d}^k_{i=1}$) are never used. So, formula 35 can be rewritten, splitting the universal quantification over $NS(s_e)$ in two universal quantification over $NS_{MV}(s_e)$ and $NS_{MV'}(s_e)$, in the following way

$$\forall s_e \in reach(M_e) \left(\begin{array}{l} \left(\bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s_e} \right) \rightarrow \\ \left(\begin{array}{l} \forall \left(\tilde{d}_{i=1}^{k} \in \tilde{D}_i, d_{j=k+1}^{n} \in D_j \right) \in NS_{MV}(s_e), \\ \exists s''_e \in next(s_e) \left[\bigwedge_{i=1}^{k} \left(\tilde{d}_i = [\![\tilde{v}_i]\!]_{s''_e} \wedge \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s''_e} \right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s''_e} \right] \wedge \\ \forall (\tilde{d}_{i=1}^{k} \in \tilde{D}_i, d_{j=k+1}^{n} \in D_j) \in NS_{MV'}(s_e), \\ \exists s'''_e \in next(s_e) \left[\bigwedge_{i=1}^{k} \left(\tilde{d}_i = [\![\tilde{v}_i]\!]_{s'''_e} \wedge \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s'''_e} \right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s'''_e} \right] \end{array} \right) \end{array} \right) \quad (36)$$

In formula 36, the matrices of the two universally quantified subformulas over $NS_{MV}(s_e)$ and $NS_{MV'}(s_e)$ are the same. According to Theorem 3, the conjunction of the two universally quantified subformulas can be replaced by a single formula universally quantified over the bigger domain $\times_{i=1}^{k} \tilde{D}_i \times \times_{j=k+1}^{n} D_j$. Note that, for any $s_e \in S_e$, $\times_{i=1}^{k} \tilde{D}_i \times \times_{j=k+1}^{n} D_j \supseteq NS_{MV}(s_e) \cup NS_{MV'}(s_e)$. This is the obtained formula:

$$\forall s_e \in reach(M_e) \left(\begin{array}{l} \left(\bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s_e} \right) \rightarrow \\ \forall d = \left(\tilde{d}_{i=1}^{k} \in \tilde{D}_i, d_{j=k+1}^{n} \in D_j \right) \\ \left(\begin{array}{l} (d \in NS_{MV}(s_e) \vee d \in NS_{MV'}(s_e)) \rightarrow \\ \exists s''_e \in next(s_e) \left[\bigwedge_{i=1}^{k} \left(\tilde{d}_i = [\![\tilde{v}_i]\!]_{s''_e} \wedge \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s''_e} \right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s''_e} \right] \end{array} \right) \end{array} \right) \quad (37)$$

Formula 37 can be rewritten, transforming the antecedent of the rightmost implication, in the following way

$$\forall s_e \in reach(M_e) \left(\begin{array}{l} \left(\bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s_e} \right) \rightarrow \\ \forall d = \left(\tilde{d}_{i=1}^{k} \in \tilde{D}_i, d_{j=k+1}^{n} \in D_j \right) \\ \left(\begin{array}{l} \exists s'_e \in next(s_e) \left[\left(\bigwedge_{i=1}^{k} \tilde{d}_i = [\![\tilde{v}_i]\!]_{s'_e} \vee \bigwedge_{i=1}^{k} \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s'_e} \right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s'_e} \right] \rightarrow \\ \exists s''_e \in next(s_e) \left[\bigwedge_{i=1}^{k} \left(\tilde{d}_i = [\![\tilde{v}_i]\!]_{s''_e} \wedge \tilde{d}_i = [\![\tilde{v}'_i]\!]_{s''_e} \right) \wedge \bigwedge_{j=k+1}^{n} d_j = [\![v_j]\!]_{s''_e} \right] \end{array} \right) \end{array} \right) \quad (38)$$

Using the *Both* and *Either* predicates (see Definition 11), formula 38 can be rewritten in the following way

$$\forall s_e \in reach(M_e) \left(\begin{array}{l} \left(\bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s_e} \right) \rightarrow \\ \forall \left(\tilde{d}_{i=1}^{k} \in \tilde{D}_i, d_{j=k+1}^{n} \in D_j \right) \left[\begin{array}{l} \exists s'_e \in next(s_e) \, [\![Either(\tilde{d}_{i=1}^{k}, d_{j=k+1}^{n})]\!]_{s'_e} \rightarrow \\ \exists s''_e \in next(s_e) \, [\![Both(\tilde{d}_{i=1}^{k}, d_{j=k+1}^{n})]\!]_{s''_e} \end{array} \right] \end{array} \right) \quad (39)$$

The inner universal quantifier can be extracted in the following way

$$\forall \left(\tilde{d}^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right),\ \forall s_e \in reach(M_e) \left(\begin{array}{l}\left(\bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s_e}\right) \rightarrow \\ \left[\begin{array}{l}\exists s'_e \in next(s_e)\ [\![Either(\tilde{d}^k_{i=1}, d^n_{j=k+1})]\!]_{s'_e} \rightarrow \\ \exists s''_e \in next(s_e)\ [\![Both(\tilde{d}^k_{i=1}, d^n_{j=k+1})]\!]_{s''_e}\end{array}\right]\end{array}\right) \quad (40)$$

Finally the two implications can be simplified in the following way[4]

$$\begin{array}{l}\forall \left(\tilde{d}^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j\right), \\ \forall s_e \in reach(M_e)\end{array} \left(\begin{array}{l}\left(\bigwedge_{i=1}^{k} [\![\tilde{v}_i]\!]_{s_e} = [\![\tilde{v}'_i]\!]_{s_e} \wedge \exists s'_e \in next(s_e)\ [\![Either(\tilde{d}^k_{i=1}, d^n_{j=k+1})]\!]_{s'_e}\right) \rightarrow \\ \exists s''_e \in next(s_e)\ [\![Both(\tilde{d}^k_{i=1}, d^n_{j=k+1})]\!]_{s''_e}\end{array}\right) \quad (41)$$

4.4.2.1 ***Checking equivalence of the transition relation in NuSMV*** In NuSMV, checking property 41 means checking that the following formula

$$\mathtt{AG}\left(\left(\left(\bigwedge_{i=1}^{k} \tilde{v}_i = \tilde{v}'_i \wedge \mathtt{EX}\left(Either\left(\tilde{d}^k_{i=1}, d^n_{j=k+1}\right)\right)\right) \rightarrow \mathtt{EX}\left(Both\left(\tilde{d}^k_{i=1}, d^n_{j=k+1}\right)\right)\right)\right) \quad (42)$$

holds in M_e, $\forall(\tilde{d}^k_{i=1} \in \tilde{D}_i, d^n_{j=k+1} \in D_j)$.

Formula 42 has been obtained from formula 41 simply applying the semantics of the CTL operators AG and EX:

- $M \models \mathtt{AG}(\varphi)$ iff $\forall s \in reach(M)\ \ ((M, s) \models \varphi)$
- $M, s \models \mathtt{EX}(\varphi)$ iff $\exists s' \in next(s)\ \ ((M, s') \models \varphi)$

Example In the following, we report some of the CTL properties that must be checked over the specification shown in Model 3 in order to prove the equivalence of the transition relations of the specifications shown in Models 1 and 2.

CTLSPEC NAME transRelOk_1 :=
AG ((amPm = amPmMut & **EX** ((AM = amPm | AM = amPmMut) & h = 0)) −>
EX ((AM = amPm & amPmMut = AM) & h = 0))
CTLSPEC NAME transRelOk_2 :=
AG ((amPm = amPmMut & **EX** ((AM = amPm | AM = amPmMut) & h = 1)) −>
EX ((AM = amPm & amPmMut = AM) & h = 1))
...
CTLSPEC NAME transRelOk_24 :=
AG ((amPm = amPmMut & **EX** ((PM = amPm | PM = amPmMut) & h = 0)) −>
EX ((PM = amPm & amPmMut = PM) & h = 0))
CTLSPEC NAME transRelOk_25 :=
AG ((amPm = amPmMut & **EX** ((PM = amPm | PM = amPmMut) & h = 1)) −>
EX ((PM = amPm & amPmMut = PM) & h = 1))
...

We must check that all the CTL properties *transRelOk_i* are true. As soon as we find a property false, we can stop checking since we will have found that the two specifications are not equivalent. In the example, we checked that all the properties *transRelOk_i*, with $i = 1, \ldots, 48$ are true. So, the two specifications are equivalent (in Section 4.4.1.1 we have also checked that they are equivalent in the initial states): the specification shown in Model 2 is an *equivalent mutant* of the specification shown in Model 1.

[4] $P \rightarrow (Q \rightarrow R) \equiv (P \wedge Q) \rightarrow R$

References

1. P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.
2. M. C. Browne, E. M. Clarke, and O. Grümberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1-2):115 – 131, 1988.
3. R. Cavada, A. Cimatti, C. A. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, and A. Tchaltsev. NuSMV 2.5 User Manual. `http://nusmv.fbk.eu/`, 2010.
4. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proceedings International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*. Springer, July 2002.
5. S. D. R. S. De Souza, J. C. Maldonado, S. C. P. F. Fabbri, and W. L. De Souza. Mutation Testing Applied to Estelle Specifications. *Software Quality Control*, 8:285–301, December 1999.
6. S. C. P. F. Fabbri, M. E. Delamaro, J. C. Maldonado, and P. C. Masiero. Mutation analysis testing for finite state machines. In *Software Reliability Engineering, 1994. Proceedings., 5th International Symposium on*, pages 220 –229, nov 1994.
7. S. C. P. F. Fabbri, J. C. Maldonado, P. C. Masiero, M. E. Delamaro, and E. Wong. Mutation Testing Applied to Validate Specifications Based on Petri Nets. In *Proceedings of the IFIP TC6 Eighth International Conference on Formal Description Techniques VIII*, pages 329–337, London, UK, 1996. Chapman & Hall, Ltd.
8. S. C. P. F. Fabbri, J. C. Maldonado, T. Sugeta, and P. C. Masiero. Mutation testing applied to validate specifications based on statecharts. In *Proceedings 10th Int Software Reliability Engineering Symp*, pages 210–219, 1999.
9. B. J. Gruen, D. Schuler, and A. Zeller. The Impact of Equivalent Mutants. In *Mutation '09: Proceedings of the 3rd International Workshop on Mutation Analysis*, pages 192–199, April 2009.
10. D. R. Kuhn. Fault Classes and Error Detection Capability of Specification-Based Testing. *ACM Transactions on Software Engineering and Methodology*, 8(4):411–424, 1999.
11. L. Liu and H. Miao. Mutation operators for Object-Z specification. In *Proceedings 10th IEEE Int. Conf. Engineering of Complex Computer Systems ICECCS 2005*, pages 498–506, 2005.
12. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.
13. The NuSMV website. `http://nusmv.fbk.eu/`, 2012.