A Note on the Stopping Failures Models

Silvio Ghilardi[1] and Silvio Ranise[2]

[1] Dipartimento di Informatica, Università degli Studi di Milano (Italy)
[2] Dipartimento di Informatica, Università di Verona (Italy)

Abstract. We investigate the introduction of the stopping failures model in order to treat universal guards in transitions of array-based systems. We conclude by some remarks explaining how the stopping failures model is implemented in the tool MCMT.

1 Introduction

This note is meant to explain how the tool MCMT deals with universal guards in transitions; the technique we study here is not specific to our approach and to our tool MCMT, on the contrary it is just a declarative realization of the *approximeted model* technique introduced in [1], [2] and implemented in systems like PFS, UNDIP. When revisiting such approximate model technique in our setting, we show that it is strictly related to a well-known topic in distributed algorithms area, namely to the *stopping failures* model [8]. In the stopping failures model, processes can crash without warning at any time; of course, a safety property is more robust if it is established for the stopping failures version of the specification of a system. As a consequence, although it is not always possible to prove a safety property for the more general stopping failures model, it is would be highly desirable to do it; surprisingly, passing to the stopping failure model can make automated verification easier and sometimes decidability is also gained in this way.

2 Notation

For general information on array-based systems the reader is referred to [5],[7], [6] We shall follow the last paper for notation and definitions, so we report here the relevant part of Section 2 of [6].

We assume the usual syntactic (e.g., signature, variable, term, atom, literal, and formula) and semantic (e.g., structure, sub-structure, truth, satisfiability, and validity) notions of first-order logic (see, e.g., [4]). The equality symbol $=$ is included in all signatures considered below. A signature is *relational* if it does not contain function symbols and it is *quasi-relational* if its function symbols are all (individual) constants. An *expression* is a term, an atom, a literal, or a formula. Let $\underline{x}$ be a finite tuple of variables and Σ a signature; a $\Sigma(\underline{x})$-expression is an expression built out of the symbols in Σ where at most the variables in $\underline{x}$ may occur free (we will write $E(\underline{x})$ to emphasize that E is a $\Sigma(\underline{x})$-expression).

According to the current practice in the SMT literature [9], a *theory* T is a pair $(\Sigma, \mathcal{C})$, where Σ is a signature and $\mathcal{C}$ is a class of Σ-structures; the structures in $\mathcal{C}$ are the *models* of T. Below, we let $T = (\Sigma, \mathcal{C})$. A Σ-formula ϕ is *T-satisfiable* if there exists a Σ-structure $\mathcal{M}$ in $\mathcal{C}$ such that ϕ is true in $\mathcal{M}$ under a suitable assignment to the free variables of ϕ (in symbols, $\mathcal{M} \models \phi$); it is *T-valid* (in symbols, $T \models \varphi$) if its negation is T-unsatisfiable. Two formulae φ_1 and φ_2 are *T-equivalent* if $\varphi_1 \leftrightarrow \varphi_2$ is *T-valid*. The *satisfiability modulo the theory T* ($SMT(T)$) *problem* amounts to establishing the T-satisfiability of quantifier-free (i.e. not containing quantifiers) Σ-formulae.

A theory $T = (\Sigma, \mathcal{C})$ is said to be *locally finite* iff Σ is finite and, for every finite set of variables $\underline{x}$, there are finitely many $\Sigma(\underline{x})$-terms $t_1, \ldots, t_{k_{\underline{x}}}$ such that for every further $\Sigma(\underline{x})$-term u, we have that $T \models u = t_i$ (for some $i \in \{1, \ldots, k_{\underline{x}}\}$). The terms $t_1, \ldots, t_{k_{\underline{x}}}$ are called $\Sigma(\underline{x})$-representative terms; if they are effectively computable from $\underline{x}$ (and t_i is computable from u), then T is said to be *effectively locally finite* (in the following, when we say 'locally finite', we in fact always mean 'effectively locally finite'). If Σ is relational or quasi-relational, then any Σ-theory T is locally finite.

A *T-partition* is a finite set $C_1(\underline{x}), \ldots, C_n(\underline{x})$ of quantifier-free formulae such that $T \models \forall \underline{x} \bigvee_{i=1}^{n} C_i(\underline{x})$ and $T \models \bigwedge_{i \neq j} \forall \underline{x} \neg (C_i(\underline{x}) \wedge C_j(\underline{x}))$. A *case-definable extension* $T' = (\Sigma', \mathcal{C}')$ of a theory $T = (\Sigma, \mathcal{C})$ is obtained from T by applying (finitely many times) the following procedure: (i) take a T-partition $C_1(\underline{x}), \ldots, C_n(\underline{x})$ together with Σ-terms $t_1(\underline{x}), \ldots, t_n(\underline{x})$;(ii) let Σ' be $\Sigma \cup \{F\}$, where F is a "fresh" function symbol (i.e. $F \notin \Sigma$) whose arity is equal to the length of $\underline{x}$; (iii) take as $\mathcal{C}'$ the class of Σ'-structures $\mathcal{M}$ whose Σ-reduct is a model of T and such that $\mathcal{M} \models \bigwedge_{i=1}^{n} \forall \underline{x}\ (C_i(\underline{x}) \rightarrow F(\underline{x}) = t_i(\underline{x}))$. Thus a case-definable extension T' of a theory T contains finitely many additional function symbols, called *case-defined functions*. It is not hard to effectively translate any $SMT(T')$ problem into an equivalent $SMT(T)$-problem, see [6] for details.

From now on, we use many-sorted first-order logic. All notions introduced above can be easily adapted to a many sorted framework. **In the rest of the paper, we fix** (i) a theory $T_I = (\Sigma_I, \mathcal{C}_I)$ for indexes whose only sort symbol is `INDEX`; (ii) a theory $T_E = (\Sigma_E, \mathcal{C}_E)$ for data whose only sort symbol is `ELEM` (the class $\mathcal{C}_E$ of models of this theory is usually a singleton). The **theory $A_I^E = (\Sigma, \mathcal{C})$ of arrays with indexes I and elements E** is obtained as the combination of T_I and T_E as follows: `INDEX`, `ELEM`, and `ARRAY` are the only sort symbols of A_I^E, the signature is $\Sigma := \Sigma_I \cup \Sigma_E \cup \{\_[\_]\}$ where $\_[\_] : \texttt{ARRAY}, \texttt{INDEX} \longrightarrow \texttt{ELEM}$ (intuitively, $a[i]$ denotes the element stored in the array a at index i); a three-sorted structure $\mathcal{M} = (\texttt{INDEX}^{\mathcal{M}}, \texttt{ELEM}^{\mathcal{M}}, \texttt{ARRAY}^{\mathcal{M}}, \mathcal{I})$ is in $\mathcal{C}$ iff $\texttt{ARRAY}^{\mathcal{M}}$ is the set of (total) functions from $\texttt{INDEX}^{\mathcal{M}}$ to $\texttt{ELEM}^{\mathcal{M}}$, the function symbol $\_[\_]$ is interpreted as function application, and $\mathcal{M}_I = (\texttt{INDEX}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_I})$, $\mathcal{M}_E = (\texttt{ELEM}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_E})$ are models of T_I and T_E, respectively (where $\mathcal{I}_{|\Sigma_X}$ is the restriction of the interpretation $\mathcal{I}$ to the symbols in Σ_X for $X \in \{I, E\}$).

Notational conventions. For the sake of brevity, we introduce the following notational conventions: d, e range over variables of sort `ELEM`, a over variables of sort `ARRAY`, $i, j, k, z, \ldots$ over variables of sort `INDEX`. An underlined variable name

abbreviates a tuple of variables of unspecified (but finite) length and, if $\underline{i} := i_1, \ldots, i_n$, the notation $a[\underline{i}]$ abbreviates the tuple of terms $a[i_1], \ldots, a[i_n]$. Possibly sub/super-scripted expressions of the form $\phi(\underline{i}, \underline{e}), \psi(\underline{i}, \underline{e})$ denote *quantifier-free $(\Sigma_I \cup \Sigma_E)$-formulae in which at most the variables $\underline{i} \cup \underline{e}$ occur* Also, $\phi(\underline{i}, \underline{t}/\underline{e})$ (or simply $\phi(\underline{i}, \underline{t})$) abbreviates the substitution of the Σ-terms $\underline{t}$ for the variables $\underline{e}$. Thus, for instance, $\phi(\underline{i}, a[\underline{i}])$ denotes the formula obtained by replacing $\underline{e}$ with $a[\underline{i}]$ in the quantifier-free formula $\phi(\underline{i}, \underline{e})$.

Special formulae. A *$\forall^I$-formula* is a formula of the form $\forall \underline{i}.\phi(\underline{i}, a[\underline{i}])$, and an *$\exists^I$-formula* is a formula of the form $\exists \underline{i}.\phi(\underline{i}, a[\underline{i}])$.

3 Array Based Systems

An *array-based (transition) system (for (T_I, T_E))* is a triple $\mathcal{S} = (a, I, \tau)$ where (i) a is the *state* variable of sort `ARRAY`;[3] (ii) $I(a)$ is the *initial* $\Sigma(a)$-formula; and (iii) $\tau(a, a')$ is the *transition* $(\Sigma \cup \Sigma_D)(a, a')$-formula, where a' is a renamed copy of a and Σ_D is a finite set of case-defined function symbols not in $\Sigma_I \cup \Sigma_E$.

Given an array-based system $\mathcal{S} = (a, I, \tau)$ and a formula $U(a)$, (an instance of) the *safety problem* is to establish whether there exists a natural number n such that the formula

$$I(a_0) \wedge \tau(a_0, a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge U(a_n) \tag{1}$$

is A_I^E-satisfiable. If there is no such n, then $\mathcal{S}$ is *safe* (w.r.t. U); otherwise, it is *unsafe* - the A_I^E-satisfiability of (1) implies the existence of a run (of length n) leading the system from a state in I to a state in U.

A general approach to solve instances of the safety problem is based on computing the set of backward reachable states. For $n \geq 0$, the *n-pre-image* of a formula $K(a)$ is $Pre^0(\tau, K) := K$ and $Pre^{n+1}(\tau, K) := Pre(\tau, Pre^n(\tau, K))$, where

$$Pre(\tau, K) := \exists a'.(\tau(a, a') \wedge K(a')). \tag{2}$$

Given $\mathcal{S} = (a, I, \tau)$ and $U(a)$, the formula $Pre^n(\tau, U)$ describes the set of backward reachable states in n steps (for $n \geq 0$). At the n-th iteration of the loop, the *basic backward reachability algorithm*, depicted in Figure 1 (a), stores in the variable B the formula $BR^n(\tau, U) := \bigvee_{i=0}^{n} Pre^i(\tau, U)$ representing the set of states which are backward reachable from the states in U in at most n steps (whereas the variable P stores the formula $Pre^n(\tau, U)$). While computing $BR^n(\tau, U)$, BReach also checks whether the system is unsafe (cf. line 3, which can be read as $I \wedge Pre^n(\tau, U)$ is A_I^E-satisfiable) or a fix-point has been reached (cf. line 2, which can be read as $\neg(BR^n(\tau, U) \rightarrow BR^{n-1}(\tau, U))$ is A_I^E-unsatisfiable or,

[3] For simplicity (and without loss of generality), we limit ourselves to array-based systems having just one variable a of sort `ARRAY`. This limitation is however dropped in the examples, where in addition T_E may be many-sorted.

```
function BReach(U)
1   P ⟵ U; B ⟵ ⊥;
2   while (P ∧ ¬B is A_I^E-sat.) do
3       if (I ∧ P is A_I^E-sat.)
                then return unsafe;
4       B ⟵ P ∨ B;
5       P ⟵ Pre(τ, P);
6   end
7   return (safe, B);
```

Fig. 1. The basic backward reachability algorithms

equivalently, that $(BR^n(\tau, U) \rightarrow BR^{n-1}(\tau, U))$ is A_I^E-valid). When BReach returns the safety of the system (cf. line 7), the variable B stores the formula describing the set of states which are backward reachable from U which is also a fix-point. Indeed, for BReach (Figure 1) to be a true (possibly non-terminating) procedure, it is mandatory that *both the A_I^E-satisfiability test for safety (line 3) and that for fix-point (line 2) are effective.* Hypotheses are needed to guarantee that.

A class of formulae whose A_I^E-satisfiability is decidable under reasonable conditions is given by the so-called $\exists^{A,I}\forall^I$-sentences, i.e. by the sentences of the kind $\exists \underline{a}\ \exists \underline{i}\ \forall \underline{j}\ \psi(\underline{i}, \underline{j}, \underline{a}[\underline{i}], \underline{a}[\underline{j}])$ for quantifier-free ψ:

Theorem 3.1 ([5]). *The A_I^E-satisfiability of $\exists^{A,I}\forall^I$-sentences is decidable if (i) T_I is locally finite and is closed under substructures; (ii) the $SMT(T_I)$ and $SMT(T_E)$ problems are decidable.*

Let us *assume conditions (i)-(ii) above from now on.* Now the problem is that of identifying a format for initial, unsafe and transition formulae so that the satisfiability tests required by the algorithm of Figure 1 falls within the $\exists^{A,I}\forall^I$-sentences covered by Theorem 3.1. In [6] the following solution is adopted to satisfy this requirement:[4]

(i) initial formulae I are assumed to be $\forall^I$*-formulae*;
(ii) formulae U describing sets of unsafe states are assumed to be $\exists^I$*-formulae*;
(iii) formulae $\tau(a, a')$ describing transitions are assumed to be in *functional form*, i.e. to be *disjunctions of* formulae of the form

$$\exists \underline{i}\, (\phi_L(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j.F_G(\underline{i}, a[\underline{i}], j, a[j]) \tag{3}$$

where ϕ_L is the *guard* (also called the local component in [5]), and F_G is a case-defined function (called the *global* component).[5]

[4] In [5] a more liberal format fulfilling the requirement is identified, however this more liberal format is not implemented in MCMT.

[5] Notice that formulae (3) can be written as first-order formulae by replacing the functional equation $a' = \lambda j.F_G(\underline{i}, a[\underline{i}], j, a[j])$ by $\forall j\, a'[j] = F_G(\underline{i}, a[\underline{i}], j, a[j])$). Notice

The reason why assumptions (ii)-(iii) work is due to the fact that one can easily show [7] that *if K is an $\exists^I$-formula and τ is in functional form, then $Pre(\tau, K)$ is equivalent to an $\exists^I$-formula.*

One can now wonder how reasonable are restrictions (i)-(ii)-(iii), i.e. to which extent they cover concrete applications. Standard benchmarks are likely to fulfill the above restrictions, however quite often the transition is a disjunction of formulae having the following more general format

$$\exists \underline{i}\, (\phi_L(\underline{i}, a[\underline{i}]) \wedge \forall k\, \psi(\underline{i}, k, a[\underline{i}], a[k]) \wedge a' = \lambda j. F_G(\underline{i}, a[\underline{i}], j, a[j])) \qquad (4)$$

In other words, formulae (4) (unlike formulae (4)) contain the universal subformula $\forall k\, \psi(\underline{i}, k, a[\underline{i}], a[k])$; let us call these transitions *universally guarded transitions in functional form*. Now the presence of such universal guards modifies the whole model-theoretic setting, undecidability arises earlier, it is not possible to prove anymore that $\exists^I$-formulae are preserved under taking preimages, and there is no evidence on how Theorem 3.1 can help.[6] Fortunately, however, universal guards can be eliminated by passing to the stopping failures version of an array-based system; this fact is a key feature of the backward search algorithms of [1], [2] and can be recaptured in our declarative setting too.

4 Stopping Failures

In a stopping failure system, a process may crash at any time without warning and without recovering. We show how one can introduce stopping failures in an array based system at a pure syntactic level.

Let $(\mathcal{S}, U)$ be a safety problem, where $\mathcal{S} = (a, I, \tau)$ is an array-based sytem and $U(a)$ is the formula describing the set of unsafe states; the stopping failures safety problem $(\tilde{\mathcal{S}}, \tilde{U})$ associated with $(\mathcal{S}, U)$ is built in few steps as follows.

(a) We first need to expand the signature to get a formula $\mathtt{C}(\mathtt{rash})(i, a[i])$ for describing the crashed processes. The easiest way to achieve this would be through the addition of a Boolean flag indicating whether a process is crashed or not. However, we work with systems having just one array variable a, so we need a slightly more complex transformation. We replace the theory T_E with a three-sorted theory $\tilde{T}_E$ whose models are the cartesian products of a model of T_E and of the two-element Boolean algebra $\{1, 0\}$. The signature Σ_E is expanded with constants $1, 0$ and with the two projections operations: this means that in the expanded signature we have unary operations pr_1 and pr_2 returning the 'old' data and the content of the 'new Boolean flag' (in the following, we write $e.1$ and $e.2$ instead of $pr_1(e)$ and $pr_2(e)$, respectively).

---

that, by abuse of notation, we still denote by A_I^E any case-definable extension of A_I^E.

[6] One may still use Theorem 3.1 for *invariant checks*, see [3] where Theorem 3.1 is proved (for the special case in which T_I is the theory of linear orders) and is used for this purpose.

Thus $\mathtt{C}(i, a[i])$ is now defined as $a[i].2 = 0$ and the complementary predicate $\mathtt{A}(i, a[i])$ is defined as $a[i].2 = 1$; we can read $\mathtt{A}(i, a[i])$ as 'the process i is active (when the system is in the state a)'. We call $\tilde{A}_I^E$ the theory resulting from the above replacement of T_E with $\tilde{T}_E$.

(b) In order to build the formulae $\tilde{I}(a)$ and $\tilde{U}(a)$ we simply take $I(a)$ and $U(a)$, replace the terms t of sort `ELEM` by $t.1$, and relativize the index quantifiers to the formula `A`. In particular, an $\exists^I$-formula $\exists \underline{i}\, \phi(\underline{i}, a[\underline{i}])$ becomes $\exists \underline{i}\, (\mathtt{A}(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{i}, a[\underline{i}].1))$ (here, if $\underline{i} = i_1, \ldots, i_n$, the abbreviation $\mathtt{A}(\underline{i}, a[\underline{i}])$ stands for $\bigwedge_{k=1}^{n} A(i_k, a[i_k])$). Similarly, a $\forall^I$-formula $\forall \underline{i}\, \phi(\underline{i}, a[\underline{i}])$ becomes $\forall \underline{i}\, (\mathtt{A}(\underline{i}, a[\underline{i}]) \rightarrow \phi(\underline{i}, a[\underline{i}].1))$.

(c) To describe the transformation of transition formulae, we first need to transform case-defined functions of sort `ELEM` having an argument j of sort `INDEX` and an argument d of sort `ELEM`. Let $F(\underline{i}, \underline{e}, j, d)$ be such a function[7] and let's its definition have the shape

$$
\begin{array}{rcl}
F(\underline{i}, \underline{e}, j, d) & := & \mathtt{case\ of}\ \ \{ \\
& & \quad C_1(\underline{i}, \underline{e}, j, d) : t_1(\underline{i}, \underline{e}, j, d) \\
& & \qquad \ldots \\
& & \quad C_k(\underline{i}, \underline{e}, j, d) : t_k(\underline{i}, \underline{e}, j, d) \\
& & \}.
\end{array}
$$

We define $\tilde{F}(\underline{i}, \underline{e}, j, d)$ by adding one case more as follows

$$
\begin{array}{rcl}
F(\underline{i}, \underline{e}, j, d) & := & \mathtt{case\ of}\ \ \{ \\
& & \quad d.2 = 0 : \langle d.1, 0 \rangle \\
& & \quad d.2 = 1 \wedge C_1(\underline{i}, \underline{e}.1, j, d.1) : \langle t_1(\underline{i}, \underline{e}.1, j, d.1), 1 \rangle \\
& & \qquad \ldots \\
& & \quad d.2 = 1 \wedge C_k(\underline{i}, \underline{e}.1, j, d.1) : \langle t_k(\underline{i}, \underline{e}.1, j, d.1), 1 \rangle \\
& & \}.
\end{array}
$$

(d) Suppose now that the transition formula $\tau(a, a')$ is built up from atomic Σ-formulae and functional equations $a' = \lambda j.F(\underline{i}, a[\underline{i}], j, a[j])$ (involving a case-defined function F) by applying to them Boolean operators as well as quantifiers of sort `ELEM` and `INDEX`.[8] In order to build $\tilde{\tau}(a, a')$ we first replace the case-defined functions F by the corresponding $\tilde{F}$, the terms t of sort `ELEM` by $t.1$, and finally we relativize all quantifiers of sort `INDEX` to the formula

[7] For simplicity and without loss of generality, we assume that case-defined function symbols are introduced at the level of the original signature Σ and not over further case-defined extensions of it.

[8] The equations $a' = \lambda j.F(\underline{i}, a[\underline{i}], j, a[j])$ can be rewritten without using λ-abstraction at first-order level, but we might not like to do that. In other words, our syntactic transformation applies to a specific syntactic form for τ and may give different (possibly not natural) results if functional equations involving case-defined functions are preprocessed and eliminated.

$\mathtt{A}$. Finally, we also add to the transition so modified, the further disjunct

$$\exists i\ (a' = \lambda j.(\mathtt{if}\ i = j\ \mathtt{then}\ \langle a[j].1, 0\rangle\ \mathtt{else}\ a[j])) \tag{5}$$

saying that processes may crash at any time.

According to the above transformations, the disjuncts (4) of the transitions in functional forms become

$$\exists \underline{i}\,(\mathtt{A}(\underline{i}, a[\underline{i}]) \wedge \phi_L(\underline{i}, a[\underline{i}].1) \wedge a' = \lambda j.\tilde{F}_G(\underline{i}, a[\underline{i}], j, a[j])) \tag{6}$$

whereas the disjuncts (4) of the universallly guarded transitions in functional forms become

$$\exists \underline{i}\,(\mathtt{A}(\underline{i}, a[\underline{i}]) \wedge \phi_L(\underline{i}, a[\underline{i}].1) \wedge \forall k\,(\mathtt{A}(k, a[k]) \rightarrow \psi(\underline{i}, k, a[\underline{i}].1, a[k].1)) \wedge a' = \lambda j.\tilde{F}_G(\underline{i}, a[\underline{i}], j, a[j])). \tag{7}$$

The intuition behind (6) and (7) is that crashed processes cannot be repaired and that only non-crashed processes can fire the transition or prevent from the transition to apply.

5 Results

We now study the relationship between a safety problem $(\mathcal{S}, U)$ and the corresponding stopping failures safety problem $(\tilde{\mathcal{S}}, \tilde{U})$. The following Proposition is easy:

Proposition 5.1. *If $\tilde{\mathcal{S}}$ is safe w.r.t. $\tilde{U}$, then $\mathcal{S}$ is safe w.r.t. U.*

Proof. Suppose (by contraposition) that

$$I(a_0) \wedge \tau(a_0, a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge U(a_n)$$

is A_I^E-satisfiable in a model $\mathcal{M}$. Consider the corresponding model $\tilde{\mathcal{M}}$ for $\tilde{A}_I^E$ and assign to the array variables a_k $(k = 0, \ldots, n)$ the function whose value at the index i is the pair formed by the previous value $a(i)$ and the Boolean value 1; let us call $\tilde{a}_k$ the arrays so defined (thus, $\tilde{\mathcal{M}} \models \bigwedge_k \forall i\, \mathtt{A}(i, \tilde{a}_k[i])$ holds). From the definitions (a)-(b)-(c)-(d) from Section 4, it is easily seen that

$$\tilde{\mathcal{M}} \models I(\tilde{a}_0) \wedge \tau(\tilde{a}_0, \tilde{a}_1) \wedge \cdots \wedge \tau(\tilde{a}_{n-1}, \tilde{a}_n) \wedge U(\tilde{a}_n)$$

holds, hence $\tilde{\mathcal{S}}$ is not safe w.r.t. $\tilde{U}$. □

Thus, a safety certification for the stopping failures version of the system implies a safety certification for the original system. Of course, the contrary is not true but we shall see that it holds for systems whose transitions are in functional form. From now on, *when we speak of an initial formula $I(a)$, we assume that I is a $\forall^I$-formula and when we speak of a safety formula $U(a)$, we assume that $U(a)$ is an $\exists^I$-formula.*

Lemma 5.2. *Suppose that $\tau(a, a')$ is in functional form, let K be an $\exists^I$-formula and let K_0 be the $\exists^I$-formula $Pre(\tau, K)$.*[9] *Then the $\exists^I$-formulae $\tilde{K} \vee \tilde{K}_0$ and $\tilde{K} \vee Pre(\tilde{\tau}, \tilde{K})$ are $\tilde{A}^I_E$-equivalent.*

Proof. Follows from (b)-(c)-(d) from Section 4 (in particular, from the fact that crashed processes cannot be recovered). □

Proposition 5.3. *If the signature Σ_I is relational and τ is in functional form, then $\tilde{\mathcal{S}}$ is safe w.r.t. $\tilde{U}$ if and only if $\mathcal{S}$ is safe w.r.t. U.*

Proof. Unsafety of $\tilde{\mathcal{S}}$ w.r.t. $\tilde{U}$ means that $\bigvee_{k \leq n} Pre^k(\tilde{\tau}, \tilde{U})$ is $\tilde{A}^E_I$-consistent with $\tilde{I}$ for some n. Now, if we put $K := \bigvee_{k \leq n} Pre^k(\tau, U)$, we have that $\tilde{K} \wedge \tilde{I}$ is $\tilde{A}^E_I$-consistent iff $K \wedge I$ is A^E_I-consistent by next Lemma. This gives the claim in view of Lemma 5.2. □

Lemma 5.4. *Suppose the signature Σ_I is relational. Let I be a $\forall^I$-formula and let K be a $\exists^I$-formula. We have that $\tilde{I} \wedge \tilde{K}$ is $\tilde{A}^E_I$-consistent iff $I \wedge K$ is A^E_I-consistent.*

Proof. One side is trivial. For the other side, suppose that $\tilde{\mathcal{M}} \models \tilde{I} \wedge \tilde{K}$ and let K be $\exists\, \underline{i} \phi(\underline{i}, a[\underline{i}])$. Thus $\tilde{\mathcal{M}} \models \tilde{I} \wedge \mathtt{A}(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{i}, a[\underline{i}])$ for some assignment to a and to the $\underline{i}$. Let the A^E_I-model $\mathcal{M}$ be obtained from the $\tilde{A}^E_I$-model $\tilde{\mathcal{M}}$ by restricting the interpretation of the sort `INDEX` to the elements assigned to the $\underline{i}$. It is clear that all processes selected for $\mathcal{M}$ are active in $\tilde{\mathcal{M}}$ (i.e. $\tilde{\mathcal{M}} \models \mathtt{A}(j, a[j])$ holds for every $j \in \mathtt{INDEX}^{\mathcal{M}}$), hence if we assign in $\mathcal{M}$ to a the array so restricted in the domain, from $\tilde{\mathcal{M}} \models \tilde{I} \wedge \phi(\underline{i}, a[\underline{i}])$, we get $\mathcal{M} \models I(a) \wedge K(a)$. □

We now come to the main point of this note: we show *how to eliminate the universal guards from the stopping failures version of a system having a universally guarded transition in functional form.* Let $\mathcal{S} = (a, I, \tau)$ be such a system and let $\tilde{\mathcal{S}} = (a, \tilde{I}, \tilde{\tau})$ be its stopping failure version (thus the disjuncts of $\tilde{\tau}$ have the form (7)). We build a third system $\mathcal{S}_+ = (a, I_+, \tau_+)$. The initial formula I_+ is $\tilde{I}$; to get τ_+, we keep (5) and the other the disjuncts (7) of $\tilde{\tau}$ are changed into

$$\exists \underline{i}\, (\mathtt{A}(\underline{i}, a[\underline{i}]) \wedge \phi_L(\underline{i}, a[\underline{i}].1) \wedge \bigwedge_{j \in \underline{i}} \psi(\underline{i}, j, a[\underline{i}].1, a[j].1) \wedge a' = \lambda j. F^+_G(\underline{i}, a[\underline{i}], j, a[j])) \tag{8}$$

[9] Recall that $\exists^I$-formulae are closed under preimages in case of of transitions in functional form, see Section 3. Modulo trivial logical manipulations, it is also clear that $\exists^I$-formulae are closed under disjunctions (the need of the disjunct $\tilde{K}$ in the statement of the Lemma is due to the fact that $\tilde{\tau}$ contains also the disjunct (5)).

where the case defined functions F_G^+ are obtained as follows. Suppose that F_G has the shape

$$\begin{aligned} F_G(\underline{i},\underline{e},j,d) \;:=\; \texttt{case of}\;\; \{ & \\ & C_1(\underline{i},\underline{e},j,d) : t_1(\underline{i},\underline{e},j,d) \\ & \quad \cdots \\ & C_k(\underline{i},\underline{e},j,d) : t_k(\underline{i},\underline{e},j,d) \\ \}; & \end{aligned}$$

we define $F_G^+(\underline{i},\underline{e},j,d)$ by adding one more case as follows

$$\begin{aligned} F_G^+(\underline{i},\underline{e},j,d) \;:=\; \texttt{case of}\;\; \{ & \\ & d.2 = 0 \vee \neg\psi(\underline{i},j,\underline{e},d.1) : \langle d.1, 0\rangle \\ & d.2 = 1 \wedge \psi(\underline{i},j,\underline{e},d.1) \wedge C_1(\underline{i},\underline{e}.1,j,d.1) : \langle t_1(\underline{i},\underline{e}.1,j,d.1), 1\rangle \\ & \quad \cdots \\ & d.2 = 1 \wedge \psi(\underline{i},j,\underline{e},d.1) \wedge C_k(\underline{i},\underline{e}.1,j,d.1) : \langle t_k(\underline{i},\underline{e}.1,j,d.1), 1\rangle \\ \}. & \end{aligned}$$

In other words, in (8) the universal guard has been removed, but processes violating it are forced to crash. The reason why $\tilde{\mathcal{S}}$ and $\mathcal{S}_+$ are equivalent is that the transition τ_+ can be simulated by finitely many failures followed by an application of $\tilde{\tau}$: in fact, the universal condition of the guard has been instantiated in (8) only to the processes firing the transition, so that the transition can fire anyway if the processes violating the universal condition of the guard crash before the transition applies. To make this intuition precise, we need a notion of bisimulation.

A *state* of A_E^I is an array s (=element of $\texttt{ARRAY}^{\mathcal{M}}$) in a finite index model $\mathcal{M}$ of A_E^I.[10] The n-th iteration of $\tau(a,a')$ is so defined: (i) $\tau^0(a,a')$ is $a = a'$; (ii) $\tau^{n+1}(a,a')$ is $\exists a''(\tau(a,a'') \wedge \tau^n(a'',a'))$.

Definition 5.5. *Suppose we are given two array-based systems $\mathcal{S}_1 = (a, I_1, \tau_1)$ and $\mathcal{S}_2 = (a, I_2, \tau_2)$ (with the same background theory A_E^I). A* bisimulation *between them is a relation R between states of A_E^I satisfying the following properties (we use the notation $\mathcal{M}_s$ to mean the finite index model where s is taken from):*

- *for every s_1 there is s_2 such that $R(s_1,s_2)$ holds (and vice versa);*
- *if $R(s_1,s_2)$ then $\mathcal{M}_{s_1} \models I_1(s_1)$ iff $\mathcal{M}_{s_2} \models I_2(s_2)$;*
- *if $R(s_1,s_2)$ and $\mathcal{M}_{s_1} \models \tau(s_1,s_1')$ then there exists s_2' and $n \geq 0$ such that $R(s_1',s_2')$ and $\mathcal{M}_{s_2} \models \tau^n(s_2,s_2')$ (and vice versa).*

The safety problems $(\mathcal{S}_1,U_1)$ and $(\mathcal{S}_2,U_2)$ are compatible with the bisimulation R iff $R(s_1,s_2)$ implies that ($\mathcal{M}_{s_1} \models U_1(s_1)$ iff $\mathcal{M}_{s_2} \models U_2(s_2)$).

The following lemma is immediately deduced from the fact that (1) is a $\exists^{A,I}\forall^I$-sentence and that an $\exists^{A,I}\forall^I$-sentence is A_I^E-satisfiable iff it is satisfiable in a finite index model (see the extended version of [5] for a proof):

[10] A finite index model $\mathcal{M}$ is a model in which the support of $\texttt{INDEX}^{\mathcal{M}}$ is a finite set.

Lemma 5.6. *Suppose that R is a bisimulation between $\mathcal{S}_1 = (a, I_1, \tau_1)$ and $\mathcal{S}_2 = (a, I_2, \tau_2)$ and that the safety problems $(\mathcal{S}_1, U_1)$ and $(\mathcal{S}_2, U_2)$ are compatible with R. Then U_1 is safe for $\mathcal{S}_1$ iff U_2 is safe for $\mathcal{S}_2$.*

Using the identity relation as a bisimulation, from the Lemma just stated and the above observations it is easy to conclude that:

Theorem 5.7. *Let $(\mathcal{S}, U)$ be a safety problem for an array-based system $\mathcal{S}$ having universally guarded transition in functional form. Then $\tilde{U}$ is safe for $\tilde{\mathcal{S}}$ iff $\tilde{U}$ is safe for the system $\mathcal{S}_+$.*

Notice that the transition of τ_+ of $\mathcal{S}_+$ is in functional form (i.e. does not have universal quantifiers in guards). We conclude the note by a couple of remarks concerning implementation details in MCMT.

Remark. MCMT does by itself the syntactic transformation from τ to τ_+. However things get a little more complicated because the $\exists^I$-formulae MCMT uses to describe backward reachable states of states are kept primitive and differentiated (see [6] to see what this means). First of all, notice that without loss of generality the disjuncts of the universally guarded transitions in functional form can be assumed to have the shape

$$\exists \underline{i}\, (\phi_L(\underline{i}, a[\underline{i}]) \wedge \forall j\, (j \notin \underline{i} \rightarrow \vartheta(\underline{i}, j, a[\underline{i}], a[j])) \wedge a' = \lambda j. F_G(\underline{i}, a[\underline{i}], j, a[j])) \quad (9)$$

where $j \in \underline{i}$ is the formula saying that j is equal to one of the $\underline{i}$'s (in fact, constraints concerning the $\underline{i}$'s can be directly inserted into ϕ_L).[11] The input format of MCMT files implicitly assumes that the quantifier $\forall j$ of the universal subformula of the guard is relative to processes not included in $\underline{i}$.[12] The formula ϑ must be given in disjunctive normal form with mutually inconsistent disjuncts[13] - these disjuncts are inserted by the user one per line following the keyword `:uguard`. When defining F_G^+, cases may be multiplied (and not just raised by 1) by MCMT, because MCMT maintains case specifications as conjunctions of literals (and not just as quantifier free formulae). In practice, MCMT does the transformation in such a way that the first cases correspond to the cases $j \in \underline{i}$, the last cases are failure cases and the intermediate cases result from the combination of one disjunct coming from ϑ and one case specification for F_G. Inconsistent cases that may arise in this way are not eliminated, but never apply because $\exists^I$-formulae arising from preimage computations must pass an SMT consistency check.

Remark. Notice that all index quantifiers in the formulae $\tilde{I}$, $\tilde{U}$ are relativized; by an argument similar to that employed for Lemma 5.2, it is easy to show

[11] The format (9) yields some simplification to the formula (8) (the conjuncts $\bigwedge_{j \in \underline{i}} \psi(\underline{i}, j, a[\underline{i}].1, a[j].1)$ become superfluous).

[12] We recall that the length of $\underline{i}$ must be at most 2 for a specification to be accepted by MCMT; in addition, the current release 1.0.1 requires such length to be 1 if the universal quantifier $\forall j$ occurs in the guard.

[13] The tool works correctly even in case the disjuncts are not mutually exclusive but it may make redundant work if this limitation is not respected.

that the backward search algorithm of Figure 1, once applied to $\mathcal{S}_+$ and $\tilde{U}$, can only produce $\exists^I$-formulae with relativized index quantifiers. Thus, MCMT in order to simplify notation and to improve readability assumes that *all the index quantifiers it works with are relativized*; as a consequence, formulae $\mathtt{A}(i, a[i])$ are never explicitly displayed (in particular, they do not occur in the report file that results from the `-r` command line option).

References

1. P. A. Abdulla, G. Delzanno, N. B. Henda, and A. Rezine. Regular model checking without transducers. In *TACAS*, volume 4424 of *LNCS*, pages 721–736, 2007.
2. P. A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite-state processes with global conditions. In *CAV*, volume 4590 of *LNCS*, pages 145–157, 2007.
3. Ahmed Bouajjani, Peter Habermehl, Yan Jurski, and Mihaela Sighireanu. Rewriting systems with data. In *Proc. of FCT 2007*, volume 4639 of *LNCS*, pages 1–22, 2007.
4. Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York-London, 1972.
5. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Towards SMT Model-Checking of Array-based Systems. In *Proc. of IJCAR*, LNCS, 2008.
6. S. Ghilardi and S. Ranise. Goal-Directed Invariant Synthesis in Model Checking Modulo Thoeries. In *Proc. of TABLEAUX 09*, LNCS, 2009. Full version available at http://homes.dsi.unimi.it/ ghilardi/allegati/GhRa-RI325-09.pdf.
7. S. Ghilardi, S. Ranise, and T. Valsecchi. Light-Weight SMT-based Model-Checking. In *Proc. of AVOCS 07-08*, ENTCS, 2008.
8. Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
9. S. Ranise and C. Tinelli. The SMT-LIB Standard: Version 1.2. Technical report, Dep. of Comp. Science, Iowa, 2006. Available at http://www.SMT-LIB.org/papers.