MODEL CHECKING WITH FLUID QUALITIES

Cristian GIUMALE[1], Mihnea MURARU[2]

This work addresses computational aspects related to a modeling approach for temporal reasoning, based on fluid qualities. *The investigated issues are the decidability and complexity of some properties of models, considered important for model checking.*

Keywords: temporal reasoning, fluid quality, evolution graph, model, behavior, model checking

1. Introduction

This work addresses computational issues related to a modeling approach based on *fluid qualities*, presented in [1]. The modeling approach aims to declaratively describe processes the evolution of which depends, in complex ways, on *time*, in a possibly non-Markovian fashion. The basic idea is that the modeled process is populated by entities, called *individuals*, which have evolving *qualities*, referred to as *fluid*, spanning various time intervals. The qualities are created, destroyed, and modified by means of *actions* applied onto individuals. Besides the individuals, qualities and actions that populate it, a *model* is also defined by its *rules*. These follow the precondition-action-effect pattern, and describe what must hold for an action to bring about its effects and what are the effects themselves. The evolution of a model is called *behavior* and is represented by a special oriented graph, named *evolution graph*, the nodes of which correspond to sets of concurrent actions, while the edges designate qualities associated to individuals.

Fig. 1 illustrates an example evolution graph. The grey nodes depict time moments and white nodes, actions. a, b and c are individuals. Initially, only the qualities $Q_1(a)$ and $Q_3(c)$ are present. When the action $a_1(a)$ is externally generated, it destroys $Q_1(a)$ and creates a new quality, $Q_2(b)$, while $Q_3(c)$ remains unaffected. A similar explanation can be given for the concurrent actions $a_2(b)$ and $a_3(c)$. Notice that there can be multiple $Q_1(a)$ qualities, within disjoint time spans.

A possible rule describing the behavior of action a_3 is given in Listing 1. `?x` and `?y` are variables, which get bound to the individuals c and a, respectively.

[1]Professor, Computer Science Department, University "Politehnica" of Bucharest, Romania, e-mail: `cristian.giumale@cs.pub.ro`

[2]Teaching assistant, Computer Science Department, University "Politehnica" of Bucharest, Romania, e-mail: `mihnea.muraru@cs.pub.ro`

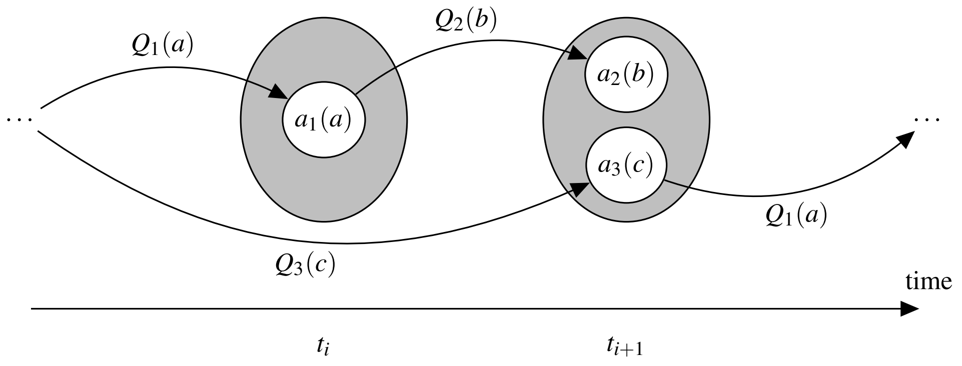

Fig. 1. Example evolution graph

`proper_part_of` is a *temporal primitive*, enforcing the restriction that the time span of the first quality must be completely included in the time span of the second.

```
rule r
    precondition    Q1(?y) proper_part_of (Q3(?x) as ?q3)
    action          a3(?x)
    postcondition   destroy ?q3, assert Q1(?y)
```

Listing 1. Possible rule describing the behavior of action a_3

The investigated issues are the decidability and the complexity of some properties of models, and are important for model checking (static and/or dynamic).

2. **Behaviors**

Let M be a model, $Actions(M)$ the finite set of actions in M, $Qualities(M)$ the finite set of qualities in M, and *Time* the set of time points, considered equipotent with the set of natural numbers.

Definition 2.1 (State). *A* state *s of M, at a given time moment t, is a subset of $Qualities(M)$, at time t.*

We use the notations $States(M) =_{\text{def}} \mathcal{P}(Qualities(M))$, and $Stimuli(M) =_{\text{def}} \mathcal{P}(Actions(M))$, where $\mathcal{P}(A)$ designates the power set of A. $States(M)$ can contain states that are never generated by M, and $Stimuli(M)$ can contain groups of actions that do not fire any rule from M. We assume that state equality is decidable.

Definition 2.2 (Behavior). *A* behavior *of a model M is a function $B : Time \to Stimuli(M) \times States(M)$, where $B(t) = (a,s)$ asserts that state s is the effect of stimulus a at time t. If the states of B depend solely on stimuli, and time is irrelevant, we say that B is* static*; otherwise, B is* fluid.

$dom(B)$ and $range(B)$ stand for the domain and range of B, respectively. If $B(t) = (a,s)$, we define two selectors: $state(B(t)) = s$ and $stimulus(B(t)) = a$.

The main relationship between a model M and a behavior $B : Time \to Stimuli(M) \times States(M)$ is that B uses the actions and qualities from M. This does not mean that B can necessarily be generated by M.

The behavior B can be represented as a time-ordered set of pairs, $B = \{(t,(a,s)) \mid t \in dom(B), (a,s) \in range(B), B(t) = (a,s)\}$. The particular representation of B is an *intensional* view of B. The *extensional* (computational) view of B relates to the effective computation of $B(t)$, which is done by a program $P_B : Time \to Stimuli(M) \times States(M)$. For a given B there are infinitely many programs that compute B. For instance, assuming that B is represented as a time-ordered, *lazy* list, containing elements from $Time \times (Stimuli(M) \times States(M))$, and that $t \in dom(B)$, the value $B(t)$ can be computed by the program:

Algorithm 2.1 Effective computation of $B(t)$

```
1:  procedure $P_B(t)$
2:      $B' \leftarrow B$
3:      while not $null(B')$ do
4:          $h \leftarrow head(B')$
5:          if $first(h) = t$ then                    ▷ $t$ corresponds to the current element
6:              return $second(h)$
7:          end if
8:          if $first(h) > t$ then                    ▷ $t$ was already skipped
9:              while true do                         ▷ Empty infinite loop
10:             end while
11:         end if
12:         $B' \leftarrow tail(B')$
13:     end while
14: end procedure
```

If L is a non-empty list, such as $\{1,2,3\}$, *head*(L) returns the first element, 1, and *tail*(L) the rest of the list, $\{2,3\}$. The components of a *lazy* list are not evaluated in advance, but only when required as the computation proceeds. If P is a pair, such as $(1,2)$, *first*(P) and *second*(P) select the two pair members, 1 and 2, respectively. By convention, if $t \notin dom(B)$, the computation of $B(t)$ does not terminate. This is enforced by the empty infinite loop, in lines 9–10.

Notice that, with a suitable representation of B, the set $dom(B)$ is *decidable*. A program can decide if t belongs to B, provided that the latter is represented as a lazy list, with elements from $Time \times (Stimuli(M) \times States(M))$, ordered increasingly according to time moments. The key here is precisely this ordering, which allows us to abandon the search for t as soon as we encounter a pair containing a later time moment.

There is no restriction on the time moments taken for probing the actions and the states of M. However, assuming that $dom(B)$ is finite and B corresponds to an evolution graph, as illustrated in Fig. 2, we can have the following interpretation of a behavior: $B = \{(t_i,(a_i,s_i)) \mid 1 \le i \le n, B(t_i) = (a_i,s_i)\}$, where t_i and t_{i+1} are consecutive time moments in the evolution graph corresponding to B (time moments when actions occur). Therefore, $\forall t \in [t_i,t_{i+1}) \bullet state(B(t)) = s_i$.

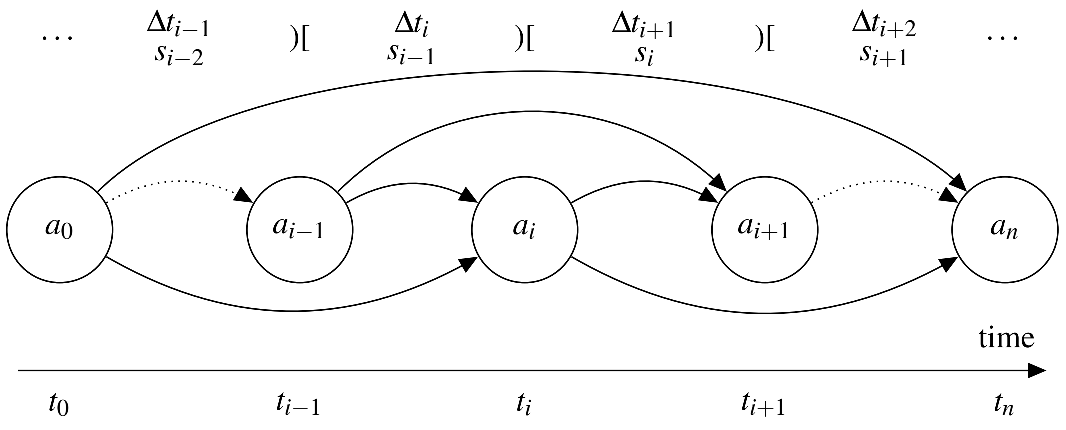

Fig. 2. A pragmatic interpretation of a behavior

3. **Models**

Definition 3.1 (Behavior sets)**.** *Let M be a model. We call $Karma(M)$ the set of all possible behaviors that are generated by M, starting from the empty behavior, B_{nil}. We call $Behaviors(M) \subsetneq \mathcal{P}(Time \times (Stimuli(M) \times States(M)))$ the set of all behaviors built with actions from $Actions(M)$ and qualities from $Qualities(M)$.*

The following relationships hold between the sets above:

(1) $Karma(M) \subseteq Behaviors(M)$.

(2) For any model M, $Karma(M)$ contains a void behavior B_{nil}, which can be represented as the empty set. The action applied on B_{nil} could be an initial action, present in any model, which sets the initial state of the model:

```
rule initBehavior
    action          initial_action
    postcondition   <assert the initial state>
```

Listing 2. Initialization rule

A model M can be viewed as an effectively computable function that computes behaviors: $M : Karma(M) \times Stimuli(M) \times Time \to Karma(M)$. For $B \in Karma(M)$ there must exist a behavior $B' \in Karma(M)$, a group of actions $a \in Stimuli(M)$, and a time t when a occurs, such that $B = M(B', a, t)$. In fact, B' is the "prefix" of B, containing all but the current entry, corresponding to stimulus a. The application $B = M(B', a, t)$ is thought as

$$dom(B) = dom(B') \setminus \{t' \in Time \mid t' > t\} \cup \{t\}$$

$$(\forall t' \in dom(B') \bullet t' < t \Rightarrow B(t') = B'(t')) \land B(t) = app(M, B', a, t),$$

where $app(M, B', a, t)$ is the result of applying the applicable rules from M, at time t, according to the previous evolution of the behavior B'. By convention, if there is no rule in M to fire for a stimulus a, then $M(B, a, t) = B$. This implicitly means that M is *total* over $Karma(M) \times Stimuli(M) \times Time$. For a usual application, the model should contain rules for all actions that can be generated by the application. Therefore, rule inapplicability could mean either that the model is faulty or incomplete or that the current actions have no effect on the current state of the model behavior.

What we want is to be able to compute any finite B in $\mathit{Karma}(M)$. Indeed, it can be proven, by induction, that, for any finite $B \in \mathit{Karma}(M)$, there is a sequence $\langle (a_1,t_1),\ldots,(a_n,t_n)\rangle$, with $a_i \in \mathit{Stimuli}(M)$ and $t_i \in \mathit{Time}$, such that $B = M(M(\ldots(M(M(B_{\text{nil}},a_1,t_1),a_2,t_2)),\ldots),a_n,t_n)$ i.e., any finite behavior from $\mathit{Karma}(M)$ can be computed by M, starting from B_{nil}. From a more general perspective, the computation of an arbitrary behavior is confronted, theoretically, with two sources of undecidability: rule applicability, and non-finiteness of the behavior time span. The former refers to looping rule preconditions, which make the interpreter unable to decide if a rule is applicable or not. However, if we consider a restricted set of models, such that rule applicability can always be decided, this restraint is removed. If no rule is applicable at a certain time moment, the model interpretation continues, with or without error messages, and the current behavior. In what follows, we adhere to this assumption. The latter points out the fact that, even if the application $M(B,a,t)$ always terminates, according to the above convention, the computation of the the entire B does not terminate, if B is not finite.

Before asserting the propositions below, we recall two definitions of set properties, which are well known in the literature [2].

Definition 3.2 (Recursive set)**.** *A set is called* recursive *if there exists a program that always terminates, and decides if an element belongs to the set. The set is also said* decidable.

Definition 3.3 (Recursively enumerable set)**.** *A set is called* recursively enumerable *if there exists a program that terminates only when a given element belongs to the set, and loops otherwise. Alternatively, there exists a* generator *for the set i.e., a program that returns a new set element on each invocation. The set is also called* semidecidable.

A recursive set is also recursively enumerable. The converse is usually false.

Proposition 3.1. *The set $\mathit{Karma}(M)$ is* semidecidable.

Proof. We show that, in the general case, $\mathit{Karma}(M)$ is recursively enumerable, without being recursive.

(a) $\mathit{Karma}(M)$ is recursively enumerable, as shown by the program GENERATE-BEHAVIOR, in Algorithm 3.1, which is a generator of behaviors of M (see Definition 3.3). The role of the **static** keyword is similar to that in C. It declares a local variable which retains its value among different calls to the procedure.

(b) $\mathit{Karma}(M)$ is *not* recursive. Assume that $\mathit{Karma}(M)$ is recursive and take $B \in \mathit{Behaviors}(M)$ randomly. By Definition 3.1, $B \in \mathit{Karma}(M)$ means that B can be computed by M, starting from B_{nil}. Then, the program TEST, in Algorithm 3.2, must terminate if $B \in \mathit{Karma}(M)$. The decision $B \in \mathit{Karma}(M)$ is reduced to deciding the termination of TEST(B,M), which is a semidecidable,

Algorithm 3.1 A generator of behaviors of a model M

```
procedure GenerateBehavior($M$)
    static $Karma \leftarrow \{B_{\text{nil}}\}$
    for $t \leftarrow 0, 1, \ldots$ do
        for-each $a \in Stimuli(M)$ do
            for-each $B \in Karma$ do
                $B' \leftarrow M(B, a, t)$
                if $B' \notin Karma$ then
                    $Karma \leftarrow Karma \cup \{B'\}$
                    return $B'$
                end if
            end for
        end for
    end for
end procedure
```

but not decidable problem. Observe that, if $dom(B)$ is not finite, the program does not terminate.

Assuming the general case, according to which any rule precondition is allowed, the undecidability of $Karma(M)$ can be proven using a reduction from the *halting problem* to the problem of deciding if a behavior belongs to $Karma(M)$. We designate the two problems as follows:

- $\text{H}\textsc{alts}(P, n)$: the problem of deciding whether the program $P : \mathbb{N} \to \mathbb{N}$ halts on input $n \in \mathbb{N}$.
- $\text{B}\textsc{elongs}(B, M)$: the problem of deciding, for a model M and a behavior $B \in Behaviors(M)$, whether $B \in Karma(M)$.

Given a program P and an input n, we must build the appropriate M and B, such that $\text{H}\textsc{alts}(P, n) \Leftrightarrow \text{B}\textsc{elongs}(B, M)$. The model M is described below:

1. The *qualities* of M are: $Qualities(M) = \{q\}$.
2. The *actions* of M are: $Actions(M) = \{a\}$.
3. The *rules* of M are:

```
1 rule r
2     precondition    not(last_action(*)), P(now()) >= 0
3     action          a
4     postcondition   q
```

Listing 3. Rule of the model M, based on the program P

Further on, we construct $B = \{(n, (\{a\}, \{q\}))\}$, which contains a single entry. Notice that it is indeed possible to choose n as a time moment, since *Time* is equipotent to $\mathbb{N}$.

The first precondition in line 2 ensures the absence of any previous action, restricting the applicability of the stimulus $\{a\}$ only to B_{nil}. Also, the second test in line 2 warrants the termination of P, when applied onto the current time

Algorithm 3.2 Deciding if $B \in \mathit{Karma}(M)$ i.e., if B can be constructed by M

```
1: procedure TEST($B$, $M$)
2:     $B' \leftarrow B_{\mathrm{nil}}$
3:     for-each $t \in dom(B)$, in increasing order do
4:         $(a, s) \leftarrow B(t)$
5:         $B' \leftarrow M(B', a, t)$
6:     end for
7:     return $B' = B$
8: end procedure
```

moment, for the rule to become active. Notice that the nonnegativity condition is a dummy test: since P produces a natural number when it terminates, the condition would be automatically satisfied. What we are actually interested in is the termination itself. If stimulus $\{a\}$ occurs at time n and $P(n)$ terminates, the quality q is asserted and state $\{q\}$ is entered, obtaining the behavior $\{(n, (\{a\}, \{q\}))\} = B$. Consequently, we have the following:

- If $P(n)$ terminates, $M(B_{\mathrm{nil}}, \{a\}, n) = B$. Hence, $B \in \mathit{Karma}(M)$.
- If $P(n)$ does not terminate, $M(B_{\mathrm{nil}}, \{a\}, n) \neq B$. Hence, $B \notin \mathit{Karma}(M)$.

Thus, HALTS(P,n) reduces to BELONGS(B,M). Had we come up with a decision procedure for BELONGS, we would have been able to use it to decide HALTS. Contradiction! Hence, $\mathit{Karma}(M)$ cannot be recursive in the general case. □

Proposition 3.2. *If the behaviors of M are* finite *then* $\mathit{Karma}(M)$ *is* decidable.

Proof. If, for all $B \in \mathit{Behaviors}(M)$, $dom(B)$ is finite, the program TEST always terminates and decides if B can be computed by M, starting from B_{nil} i.e., if $B \in \mathit{Karma}(M)$. □

4. Properties

From all the behaviors in $\mathit{Behaviors}(M)$, only a part could be of interest according to the modeled application. The focus of interest can be specified as a property, say *Prop*, which, if nontrivial, splits $\mathit{Behaviors}(M)$ into nonempty sets containing behaviors that satisfy *Prop* and behaviors that do not satisfy *Prop*, respectively. In a particular case, the truth set of *Prop* could be a restricted subset of $\mathit{Karma}(M)$.

Definition 4.1 (Property). *A* property *of a model M is a function* $\mathit{Prop} : \mathit{Behaviors}(M) \times \mathit{States}(M) \times \mathit{Time} \to \mathit{Bool}$.

Definition 4.2 (Plausibility). *The concept is defined with respect to both states and behaviors, as follows:*

(a) *The* plausibility *of a* state $s \in \mathit{States}(M)$, *at a moment* $t \in \mathit{Time}$, *according to the property Prop and a behavior* $B \in \mathit{Behaviors}(M)$, *is:*

$$\mathit{PlausibleState}(s, B, t, \mathit{Prop}) =_{def} t \in dom(B) \wedge s = state(B(t)) \wedge \mathit{Prop}(B, s, t).$$

(b) The plausibility *of a* behavior $B \in Behaviors(M)$*, according to the property Prop, is:*

$$Plausible(B, Prop) =_{def} \forall t \in dom(B) \bullet PlausibleState(state(B(t)), B, t, Prop)$$
$$= \forall t \in dom(B) \bullet Prop(B, state(B(t)), t).$$

Notice that, since $dom(B_{\mathrm{nil}}) = \emptyset$, we have that $\forall Prop \bullet Plausible(B_{\mathrm{nil}}, Prop)$.

Definition 4.3 (Realizability)**.** *The concept is defined with respect to both states and properties, as follows:*

(a) The realizability *of a* state $s \in States(M)$*, according to the property Prop, is:*

$$Realizable(s, Prop) =_{def} \exists B \in Behaviors(M) \bullet$$
$$(Plausible(B, Prop) \wedge \exists p \in range(B) \bullet s = state(p)).$$

(b) The realizability *of a* property *Prop is:*

$$Realizable(Prop) =_{def} \exists B \in Behaviors(M) \bullet B \neq B_{\mathrm{nil}} \wedge Plausible(B, Prop).$$

Definition 4.4 (Extension)**.** *The* extension *of a property Prop of a model M is the set*

$$Ext(Prop) =_{def} \{B \in Behaviors(M) \mid Plausible(B, Prop)\}.$$

Since $\forall Prop \bullet Plausible(B_{\mathrm{nil}}, Prop)$, we have that $\forall Prop \bullet Ext(Prop) \neq \emptyset$. In some cases, a property can focus on behaviors from $Karma(M)$ only, as demonstrated by $Prop(B, s, t) = B \in Karma(M) \wedge Prop'(B, s, t)$. In other cases, it is useful to consider *Prop* total over $Behaviors(M)$. Such a case addresses the *correctness* of a model.

Definition 4.5 (Correctness)**.** *The concept is applied to models and has two flavors, strong and weak, as described below.*

(a) A model M is strongly correct*, according to a property* $Prop : Behaviors(M) \times States(M) \times Time \to Bool$*, if* $Karma(M) \subseteq Ext(Prop)$*.*

(b) A model M is weakly correct*, according to a property Prop, if* $Karma(M) = \{B_{\mathrm{nil}}\} \vee Karma(M) \cap Ext(Prop) \neq \{B_{\mathrm{nil}}\}$*. See the alternative Definition 5.2 for more details.*

Definition 4.6 (Triviality and extensionality)**.** *A property Prop of a model M is:*

(a) Trivial*, if* $Ext(Prop) = Behaviors(M)$*.*

(b) Extensional*, if, for any* $B \in Behaviors(M)$*, the decision* $B \in Ext(Prop)$ *does not depend on the program used to compute B. More precisely,*

$$\forall B \in Behaviors(M) \bullet \forall P, R \in Programs(B) \bullet$$
$$P = R \Rightarrow (B \in_{using\ P} Ext(Prop) \Leftrightarrow B \in_{using\ R} Ext(Prop)).$$

$P = R$ designates the *computational equivalence* of the two programs: for the same input, either both programs produce the same output or both fail to terminate.

Proposition 4.1. *A nontrivial and extensional property Prop of a model is not decidable i.e., the set* $Ext(Prop)$ *is not decidable. The proposition is a particularization of Rice's theorem* [3].

Proof. Let *Prop* be a nontrivial and extensional property of a model M, and assume that *Prop* is decidable. Consider a non-terminating program $P_{B_{\text{nil}}}$, which effectively computes the B_{nil} behavior i.e., the totally undefined function in $Hom(Time, Stimuli(M) \times States(M))$, where $Hom(A,B)$ designates the set of functions from A to B. Since *Prop* is nontrivial, it splits $Behaviors(M)$ into two nonempty subsets, one corresponding to $Ext(Prop)$, the other to $\overline{Ext}(Prop) =_{\text{def}} Behaviors(M) \setminus Ext(Prop)$. $Ext(Prop)$ cannot be empty, since it always contains B_{nil}; $\overline{Ext}(Prop)$ cannot be empty either since, otherwise, $Ext(Prop) = Behaviors(M)$ and *Prop* would be trivial.

Let B be a behavior from $\overline{Ext}(Prop)$. Certainly, we have $B \neq B_{\text{nil}}$, since $B_{\text{nil}} \in Ext(Prop)$. Choose, at random, a behavior $B' \in Behaviors(M)$ and $t' \in Time$ and build the program:

Algorithm 4.1 Program that computes the behavior B''

```
1: procedure R($t$)
2:     $P_{B'}(t')$
3:     return $P_B(t)$
4: end procedure
```

where $P_{B'}$ and P_B are programs that compute B' and B, respectively. We have the following:

- If $t' \notin dom(B')$ then $P_{B'}(t')$ does not terminate and $R = P_{B_{\text{nil}}}$.
- If $t' \in dom(B')$ then $P_{B'}(t')$ terminates and $R = P_B$.

Therefore, the program R computes a behavior B'':

- If $P_{B'}(t')$ does not terminate, $B'' = B_{\text{nil}}$. Hence, by the extensionality of *Prop*, we have that $B'' \in Ext(Prop)$.
- If $P_{B'}(t')$ terminates, $B'' = B$. Hence, by the extensionality of *Prop*, we have that $B'' \notin Ext(Prop)$.

Thus, deciding if $P_{B'}(t')$ terminates reduces to deciding if $B'' \in Ext(Prop)$. Hence, *Prop* cannot be decidable, in the general case. □

Proposition 4.2. *Let M be a model with behaviors spanning a* finite *time interval Δt, and Prop a property of M. Prop is* decidable.

Proof. Let us first note that "a model with finite behaviors" is different from "a model with behaviors over a finite time interval". If the model M has behaviors over a finite time interval then M has finite behaviors; the reciprocal is not true.

Since the time interval is finite, each behavior of M is finite. Assume that n is the number of time points in Δt (the moments when actions occur), and that m is the number of actions in M. Then, the number of all possible distinct groups of actions (the number of distinct nodes that can be part of an evolution graph) is

$$ag = \sum_{i=1}^{m} \binom{m}{i} = 2^m - 1.$$

The sum starts from 1, as empty groups of actions are not taken into account. A behavior B of M with $|dom(B)| = i$, $1 \leq i \leq n$, corresponds to a permutation of size

i, with repetitions, containing stimuli (groups of actions) from a set of ag elements. $|A|$ denotes the number of elements in set A. The overall number of behaviors of M, with length i, is $ag^i = (2^m - 1)^i < 2^{mi}$. Thus,

$$|Karma(M)| < \sum_{i=1}^{n} 2^{mi} < \frac{1-(2^m)^n}{1-2^m}.$$

Assume that the verification of *Prop*, at any moment $t \in \Delta t$, takes at most c time units. Then, verifying a behavior of length n takes, in the worst case, $c(1+2+\ldots+n) = O(n^2)$. This is due the fact that, while the behavior unfolds, the property is checked at each time $t \in [0,n]$, and for all intermediate time points $t' \in [0,t]$. Verifying all the behaviors of M takes $O(n^2|Karma(M)|)$ time units, a finite time. □

Proposition 4.3. *Let* REALIZABLEPROP$(M, Prop)$ *be the problem "For a model M, with behaviors spanning a finite time interval, and a property Prop, decide whether Realizable(Prop)".* REALIZABLEPROP *is NP-hard.*

Proof. We build a polynomial-time reduction SAT $\leq_p$ REALIZABLEPROP, where SAT is the well-known boolean satisfiability problem.

Let $F = T_1 \wedge T_2 \wedge \ldots \wedge T_n$ be a formula in conjunctive normal form propositional calculus, $Var(F)$ the variables from F, and $m = |Var(F)|$. Starting from the formula F, we build a model M_F, in the following way:

(1) The *qualities* of M_F are: $Qualities(M_F) = \bigcup_{v \in Var(F)} \{q_v\}$. The presence of q_v at time t in a behavior B of M_F means $v = 1$ at time t and during the time-slice associated to q_v. The absence of q_v at time t in a behavior B of M_F means $v = 0$ at time moment t.

(2) The *actions* of M_F are: $Actions(M_F) = \{\texttt{initial\_action}\} \cup \bigcup_{v \in Var(F)} \{a_{v,1}, a_{v,0}\}$. The occurrence of $a_{v,1}$ asserts the quality q_v i.e., it sets $v = 1$. The occurrence of $a_{v,0}$ destroys the quality q_v i.e., it sets $v = 0$.

(3) The *rules* of M_F are given below. For each $v \in Var(F)$ there are two rules: `v_is_1` and `v_is_0`. `[now(), now()]` designates the unit time interval, containing only the current moment.

```
rule v_is_1          -- One for each variable v. Sets v = 1.
    precondition     not ([now(), now()] proper_part_of qv)
    action           av,1
    postcondition    qv

rule v_is_0          -- One for each variable v. Sets v = 0.
    precondition     [now(), now()] proper_part_of qv
    action           av,0
    postcondition    destroy qv

rule doNothing       -- Fires at any t in [0, n]
    precondition     now() <= n
    action           initial_action
```

```

rule stop
    precondition    now() > n
    action          initial_action
    postcondition   halt
```

Listing 4. Rules of the model M_F, based on the formula F

Since `initial_action` is present at any time in the model environment, M_F has at least an applicable rule at any moment t in the interval $[0,n]$. Therefore, each behavior $B \in \mathit{Karma}(M_F)$ contains all the intermediate time points $1,2,\ldots,t-1$. The model unfolds from time 0 to time n, starting from B_{nil}, such that, for any $B \in \mathit{Karma}(M_F)$, $\mathit{dom}(B) = [0,t]$, with $t \leq n$. At time 0, there are no qualities in the unfolding behavior. Hence, initially, the variables from $\mathit{Var}(F)$ are all bound to 0.

Let B be a behavior in $\mathit{Karma}(M_F)$. At time $t \in \mathit{dom}(B)$, the state $\mathit{state}(B(t))$ is a subset of $\mathit{Qualities}(M_F)$ and corresponds to particular bindings of the variables from $\mathit{Var}(F)$, which eventually make $T_i = 1$, for some $i \in [1,n]$, as shown in Fig. 3.

We define the property Prop_F as follows:

$$\mathit{Prop}_F(B,s,t) =_{\mathrm{def}} t > 0 \Rightarrow \bigwedge_{i=1}^{t} \left(\bigvee_{v \in T_i} q_v \in s \quad \vee \quad \bigvee_{\neg v \in T_i} q_v \notin s \right).$$

Consequently, we have:

$$\mathit{Realizable}(\mathit{Prop}_F) = \exists B \in \mathit{Behaviors}(M) \bullet B \neq B_{\mathrm{nil}} \wedge \mathit{Plausible}(B, \mathit{Prop}_F),$$

where

$$\begin{aligned}
&\mathit{Plausible}(B, \mathit{Prop}_F) \\
&\quad = \forall t \in \mathit{dom}(B) \bullet \mathit{Prop}_F(B, \mathit{state}(B(t)), t) \\
&\quad = \forall t \in \mathit{dom}(B) \bullet t > 0 \Rightarrow \bigwedge_{i=1}^{t} \left(\bigvee_{v \in T_i} q_v \in \mathit{state}(B(t)) \vee \bigvee_{\neg v \in T_i} q_v \notin \mathit{state}(B(t)) \right).
\end{aligned}$$

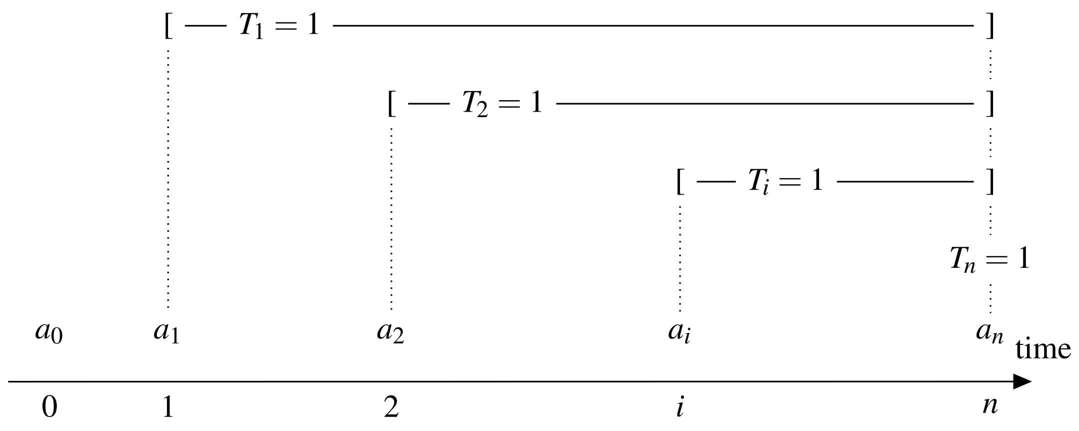

Fig. 3. A behavior of M_F such that $\mathrm{SAT}(F)$ is true

Observe that $q_v \in state(B(t))$ means $v = 1$, and $q_v \notin state(B(t))$ means $v = 0$. Then, $Plausible(B, Prop_F)$ means that, according to the behavior B, at any time $t \in dom(B)$, with $t > 0$, $T_i = 1$, for $1 \leq i \leq t$. Hence, if $n \in dom(B)$, then $Plausible(B, Prop_F)$ means $\mathrm{SAT}(F)$ at time n, according to the behavior B. Therefore, $Realizable(Prop_F) \Rightarrow \mathrm{SAT}(F)$.

Conversely, if $\mathrm{SAT}(F)$, there must be some bindings *BVar* of variables from $Var(F)$ such that $T_i = 1$, for $1 \leq i \leq n$. Imagine that these bindings are performed incrementally over a time interval $[0, n]$, such that $T_i = 1$ at any time $t \in [i, n]$, for $1 \leq i \leq n$. Therefore, there must be a behavior B, with $dom(B) = [0, n]$, such that $state(B(n))$ contains qualities corresponding to the bindings from *BVar*. It is easy to see that $Realizable(Prop_F)$ holds. Hence, $\mathrm{SAT}(F) \Rightarrow Realizable(Prop_F)$.

Notice that the model M_F and the property $Prop_F$ can be built in polynomial time, function of m and n. Let *Constr* be a tractable program i.e., with polynomial complexity, such that the application $Constr(F)$ returns the pair $(M_F, Prop_F)$. We have that

$$\mathrm{SAT}(F) \Leftrightarrow \textsc{RealizableProp}(Constr(F)).$$

Since all the behaviors of M_F span the time interval $[0, n]$ and, by Proposition 4.2, $Prop_F$ is decidable, the problem REALIZABLEPROP can be effectively solved i.e., there exists a program that solves the problem.

Since F has been selected randomly, it follows that $\mathrm{SAT} \leq_p$ REALIZABLEPROP i.e., REALIZABLEPROP is NP-hard. □

Proposition 4.4. *Let M be a model with behaviors spanning a finite time interval, and $Prop : Behaviors(M) \times States(M) \times Time \to Bool$ a property of M. If Prop can be decided tractably, then* $\textsc{RealizableProp}(M, Prop)$ *is NP-complete.*

Proof. Consider that M has finite behaviors that span a time interval $[0, n]$. Then, the set $Behaviors(M)$ is finite. Assume that the complexity of deciding *Prop* is $O(P(n))$, were $P(n)$ is a polynomial in n, and build the nondeterministic Algorithm 4.2.

Algorithm 4.2 Nondeterministic algorithm which solves REALIZABLEPROP

```
1: procedure TEST($M$)
2:     $B \leftarrow$ choice($Behaviors(M)$)
3:     for-each $t \in dom(B)$ do
4:         if $\neg Prop(B, state(B(t)), t)$ then
5:             fail
6:         end if
7:     end for
8:     success
9: end procedure
```

The nondeterministic algorithm employs the following primitives [2]. **choice** takes a set as an argument, and splits the execution of the nondeterministic algorithm into a numbers of threads equal to the number of elements in the set, returning a different element on each thread. **fail** terminates the current thread only, while **success** terminates the entire algorithm. The algorithm succeeds if *at least one* thread succeeds, and fails if *all* threads fail. All three operations have $O(1)$ complexity.

It is easy to see that TEST solves REALIZABLEPROP. Since the angelic complexity of TEST is $O(nP(n))$, it follows that REALIZABLEPROP is in NP. By angelic complexity [2], we understand the total complexity of the operations on the shortest path that leads to success. Since, according to Proposition 4.3, REALIZABLEPROP is NP-hard, it follows that REALIZABLEPROP is NP-complete. □

5. **Model Checking**

According to Definitions 4.2–4.3, we can define the following predicates, where M is a model, $S \subseteq \mathit{Behaviors}(M)$, $S \neq \emptyset$, and $\mathit{Prop} : \mathit{Behaviors}(M) \times \mathit{States}(M) \times \mathit{Time} \rightarrow \mathit{Bool}$ is a property of M.

Definition 5.1 (Satisfiability and realizability)**.** *The* satisfiability *and* realizability *of a property Prop, with respect to a set of behaviors S, are:*

$$\mathit{Satisfiable}(\mathit{Prop}, S) =_{\mathit{def}} \forall B \in S \bullet \mathit{Plausible}(B, \mathit{Prop})$$
$$\mathit{Realizable}(\mathit{Prop}, S) =_{\mathit{def}} \exists B \in S \bullet \mathit{Plausible}(B, \mathit{Prop}).$$

The following relation holds trivially:

$$\mathit{Satisfiable}(\mathit{Prop}, S) \Rightarrow \mathit{Realizable}(\mathit{Prop}, S).$$

Proposition 5.1. *Let* FOCALIZEDREALIZABLEPROP(M, Prop, S) *be the problem "For a model M, with behaviors* $S \subseteq \mathit{Behaviors}(M)$ *spanning a finite time interval, and a property Prop, decide whether* $\mathit{Realizable}(\mathit{Prop}, S)$*".* FOCALIZEDREALIZABLEPROP *is NP-hard.*

Proof. The proof is similar to that of Proposition 4.3, by taking S instead of $\mathit{Behaviors}(M)$. □

Proposition 5.2. *Let M be a model,* $S \subseteq \mathit{Behaviors}(M)$ *a set of behaviors spanning a finite time interval, and* $\mathit{Prop} : \mathit{Behaviors}(M) \times \mathit{States}(M) \times \mathit{Time} \rightarrow \mathit{Bool}$ *a property of M. If Prop can be decided tractably, then* FOCALIZEDREALIZABLEPROP(M, Prop, S) *is NP-complete.*

Proof. The proof is similar to that of Proposition 4.4. □

Definition 5.2 (Correctness)**.** *Let M be a model, and* $\mathit{Prop} : \mathit{Behaviors}(M) \times \mathit{States}(M) \times \mathit{Time} \rightarrow \mathit{Bool}$ *a property of M.*

(a) We say that M is strongly correct*, according to Prop, if:*

$$\mathit{StronglyCorrect}(M, \mathit{Prop}) =_{\mathit{def}} \mathit{Satisfiable}(\mathit{Prop}, \mathit{Karma}(M)).$$

(b) We say that M is weakly correct*, according to Prop, if:*

$$\mathit{WeaklyCorrect}(M, \mathit{Prop}) =_{\mathit{def}}$$
$$\mathit{Karma}(M) = \{B_{\mathrm{nil}}\} \vee \mathit{Realizable}(\mathit{Prop}, \mathit{Karma}(M) \setminus \{B_{\mathrm{nil}}\}).$$

It is easy to see that Definitions 4.5 and 5.2 are equivalent.

Proposition 5.3. *We have that:*

$$StronglyCorrect(M, Prop) \Rightarrow WeaklyCorrect(M, Prop).$$

Proof. We distinguish two cases:

(a) If $Karma(M) = \{B_{\text{nil}}\}$, we have that:

$$\begin{aligned} StronglyCorrect(M, Prop) &=_{\text{def}} Satisfiable(Prop, Karma(M)) \\ &= Satisfiable(Prop, \{B_{\text{nil}}\}) = Plausible(B_{\text{nil}}, Prop) = 1 \\ WeaklyCorrect(M, Prop) &=_{\text{def}} Karma(M) = \{B_{\text{nil}}\} \vee \ldots = 1. \end{aligned}$$

Thus, the proposition becomes $1 \Rightarrow 1$.

(b) If $Karma(M) \neq \{B_{\text{nil}}\}$, we have that:

$$\begin{aligned} &StronglyCorrect(M, Prop) \\ &\quad =_{\text{def}} Satisfiable(Prop, Karma(M)) \Rightarrow \exists B \in Karma(M) \setminus \{B_{\text{nil}}\} \bullet Plausible(B, Prop) \\ &\quad \Rightarrow Realizable(Prop, Karma(M) \setminus \{B_{\text{nil}}\}) \Rightarrow WeaklyCorrect(M, Prop). \end{aligned}$$

□

Definition 5.3 (Static checking)**.** *The concept has two flavors, strong and weak:*

(a) The weak static checking *of a model M, with respect to a property Prop, means deciding WeaklyCorrect(M, Prop).*

(b) The strong static checking *of a model M, with respect to a property Prop, means deciding StronglyCorrect(M, Prop).*

Proposition 5.4. *Let M be a model, and Prop a property of M.*

(a) If the behaviors of M span a finite time interval, then the weak static checking of M, with respect to Prop, is NP-hard.

(b) If the behaviors of M span a finite time interval, and Prop can be decided tractably, then the weak static checking of M is NP-complete.

Proof. The facts derive trivially from Propositions 5.1 and 5.2, by taking $S = Karma(M) \setminus \{B_{\text{nil}}\}$. □

Proposition 5.5. *Let M be a model, and Prop a property of M. If the behaviors of M span a finite time interval, and Prop can be decided tractably, then the strong static checking of M, with respect to Prop, is in NP.*

Proof. Consider that M has finite behaviors that span a time interval $[0, n]$. Then, the set $Karma(M)$ is finite. Assume that the complexity of deciding $Prop$ is $O(P(n))$, were $P(n)$ is a polynomial in n, and build the nondeterministic Algorithm 5.1. Since the angelic complexity of TEST is $O(nP(n))$, it follows that the strong static checking is in NP. □

Proposition 5.6. *Let M be a model, and Prop a property of M. If the behaviors of M span a finite time interval, and Prop cannot be decided tractably, then the strong static checking of M, with respect to Prop, is not in NP.*

Algorithm 5.1 Nondeterministic algorithm for strong static checking

```
procedure STRONGLYCORRECT($M, Prop$)
    return $\neg$TEST($M, Prop$)
end procedure

procedure TEST($M, Prop$)
    $B \leftarrow$ choice($Karma(M) \setminus \{B_{\mathrm{nil}}\}$)
    for-each $t \in dom(B)$ do
        if $\neg Prop(B, state(B(t)), t)$ then
            success
        end if
    end for
    fail
end procedure
```

Proof. The angelic complexity of STRONGLYCORRECT in Algorithm 5.1 is supra-polynomial in n. Moreover, the algorithm adds to the complexity of the model checking a fringe polynomial complexity only i.e., there is no "faster" strong static checking algorithm. □

Definition 5.4 (Dynamic correctness)**.** *Let M be a model, Prop a property of M, and $B \in Karma(M)$ a behavior that can be generated by M. We say that M is* dynamically correct*, with respect to B and Prop, if $Prop(B, state(B(t)), t)$ holds for any $t \in dom(B)$, while the behavior B unfolds:*

$$DynamicallyCorrect(M, B, Prop) =_{def} Plausible(B, Prop).$$

Definition 5.5 (Dynamic checking)**.** *Let M be a model, Prop a property of M, and $B \in Karma(M)$ a behavior that can be generated by M. We call* dynamic checking *of M the process of deciding whether M is dynamically correct, with respect to B and Prop.*

Proposition 5.7. *Let M be a model, and Prop a property of M. If Prop can be decided tractably, and M has behaviors spanning a finite time interval, then the dynamic checking of M is tractable.*

Proof. Assume that the current unfolding behavior, B, has $dom(B) = [0, n]$. Moreover, consider that testing $Prop(B, s, t)$ has the complexity $O(n^c)$, where c is a positive constant. In the worst case, at time $t \in [0, n]$, the property *Prop* is tested for $B(0), B(1), \ldots, B(t)$. Therefore, the time spent is $tO(n^c)$. The overall time spent for testing B is:

$$\sum_{t=0}^{n} tO(n^c) = O(n^{c+2}).$$ □

6. Conclusions

Proposition 4.1 applies to the static model checking without any restriction on the finiteness of model behaviors. Proposition 4.2 applies to the static checking of models with behaviors over a finite time interval, as usually encountered in practice. Propositions 5.4, 5.5 and 5.6 show that there is no foreseeable hope for the tractable static checking of a model, unless restrictions are enforced on the properties.

Unfortunately, even dynamic checking means investigating incrementally behaviors that span a finite time interval, which grows during the model interpretation. Therefore, in the general case, the dynamic checking of model behavior complies to Proposition 4.3. A possible solution is to encode the checked property, *Prop*, directly in the model rules, such that the behavior is guaranteed to satisfy *Prop*. Nevertheless, the application of rules will implicitly do the part required for checking *Prop*, according to the current state of the evolution graph. The process is similar to the explicit checking of *Prop*. Hence, even this solution is NP-hard. The hope is that the complexity of the dynamic checking of a property *Prop* could be reduced if the checking is focused on the recent past of the evolving behavior, or if the checking process can preserve its status from one model interpretation cycle to the next. A trivial way out, that complies to Proposition 5.7, is to hope for a tractable property used for the dynamic model checking.

Some of the observations above address the ambitious endeavor of checking whether a property *Prop* of a model M is realizable according to $Karma(M)$. In other words, we are trying to answer the question "Could M generate a behavior B that is plausible according to *Prop*?", which means to *predict* the evolution of M. This kind of processing is required in an application where the model is used to control the current state of the application in function of both the *past* evolution and of what might happen in the *future*. The model interpreter has to unfold plausible future behaviors and choose a particular "application-suitable" one as the way for extending the current model behavior.

Acknowledgement

The work has been funded by the Sectorial Operational Programme Human Resources Development 2007–2013, of the Romanian Ministry of Labour, Family and Equal Opportunities, through the Financial Agreement POSDRU/88/1.5/S/61178.

REFERENCES

[1] *C. Giumale, L. Negreanu, M. Muraru, M. Popovici, A. Agache and C. Dobre*, "Modeling with fluid qualities", Proc. of the 18th International Conference on Control Systems and Computer Science (CSCS-18), 2011, pp. 693–698.

[2] *C. Giumale*, "Introducere în Analiza Algoritmilor: Teorie şi Aplicaţie" ("Introduction to Analysis of Algorithms: Theory and Applications"), Polirom, Iaşi, România, 2004.

[3] *H. G. Rice*, "Classes of recursively enumerable sets and their decision problems", Trans. Amer. Math. Soc., **vol. 74**, 1953, pp. 358–366.