Observations on the Decidability of Transitions*

Yuri Gurevich[1] and Rostislav Yavorskiy[2]**

[1] Microsoft Research
Redmond WA, 98052, USA
gurevich@microsoft.com
[2] Steklov Mathematical Institute
Gubkina 8, Moscow, 119991, Russia
rey@mi.ras.ru

Abstract. Consider a multiple-agent transition system such that, for some basic types $T_1, \ldots, T_n$, the state of any agent can be represented as an element of the Cartesian product $T_1 \times \cdots \times T_n$. The system evolves by means of global steps. During such a step, new agents may be created and some existing agents may be updated or removed, but the total number of created, updated and removed agents is uniformly bounded.
We show that, under appropriate conditions, there is an algorithm for deciding assume-guarantee properties of one-step computations. The result can be used for automatic invariant verification as well as for finite state approximation of the system in the context of test-case generation from AsmL specifications.

1 Motivating example

Consider the following simplified model of a file system (a real world file system that the authors were exposed to). Basic information about a file is collected in the `File` class. For simplicity, we include in the model only very basic file attributes: name and sort. Also, each file keeps a set of the identifiers of its children, and a reference to the parent. Suppose that we want to verify that these references are mutually consistent, that is that every child knows its parent and that every parent knows all the children. We use the syntax of the Abstract State Machines specification language [1, 2].

```
type FileId = Integer

enum FileAttr
  Regular
  Directory

class File
  var fid as FileId // explicit unique identifier
```

* Proc. ASM 2004, Springer Lecture Notes in Computer Science
** The second author is partially supported by grants from Russian Foundation for Basic Research, Russian Science Support Foundation, and Microsoft Research.

```
  var sort as FileAttr
  var name as String // file name
  var parent as FileId //reference to the parent
  var children as Set of FileId // the children
  var content as String // the children
```

Thus a file system is modeled as a set of file records. The `fid` field provides an explicit unique identifier of a file. As an object (or class instance), a file is automatically provided with an object ID, but there may be good reasons to have explicit identifiers as well.

The root of the file system has a file ID but no other file fields. A global variable `files` contains the current set of existing files; initially it is empty. And there is a counter that provides fresh file IDs.

```
const root as FileId = 0
var files as Set of File = {}
var nextFid as FileId = 1
```

We specify common file system operations for creation, deletion and renaming/moving of a resource.

```
procedure Create(parent as File, name as String, sort as FileAttr)
  let fid = nextFid
  let file = new File(fid, sort, name, parent,{},"")
  add file to files
  add fid to parent.children
  nextFid += 1

procedure Delete(file as File)
  let parent = the file′ in files where file.parent = file′.fid
  require file.children = {}
  remove file.fid from parent.children
  remove file from files

procedure Move (file as File, newName as String, newParent as File)
  file.parent := newParent.fid
  file.name := newName
  add file.fid to newParent.children
  let oldParent = the file' in files where file.parent = file'.fid
  remove file.fid from oldParent.children
```

The following method formalizes the property to be verified.

```
Invariant() as Boolean
  forall f1 in files, f2 in files holds
    (f1 ne f2) implies (f1.fid in f2.children iff f1.parent = f2.fid)
```

Note that the operations `Create` and `Delete` affect two files — all the rest remain unchanged. The `Move` operation affects only three files, namely the moved file, its old parent and its new parent).

A “manual” proof of the property is simple. Assume that the invariant holds before a transition. One only needs to check that, after the transition is performed, the property holds for the affected files. This splits into several cases. In each cases, checking of the property is easy.

Similar examples arise in modelling of various distributed systems. Usually, in multiple agent transition systems the state of an agent a is characterized by the values of fields $a.f_1, \ldots, a.f_N$. Without loss of generality one may assume that all agents have the same set of fields. During one step of the computation some new agents may be created and some previously active agents may be removed or updated. The main restriction we impose is that the set of affected agents — created, removed or updated — is uniformly bounded for all steps.

In the example we checked an invariant. More generally, we may want to check whether a precondition φ_1 at a given state of the system implies a postcondition φ_2 at the next state of the system.

We sought (and found) a decidability result that covers invariant checking and assume-guarantee properties for such systems. The result can be used for automatic invariant verification as well as for finite state approximation of the system in the context of test-case generation from AsmL specifications [3, 4].

The rest of the paper is organized as follows. In Section 2 we describe our computation model. In Section 3 we show that under the considered restrictions the assume-guarantee properties of a system are expressible in the first order theory of the underlying structure S. The conclusion is given in Section 4.

2 Computational model

2.1 The system state

Let I denote an infinite index set, that is any countable set with only the equality relation defined on the elements. For simplicity we may assume I is the set of natural numbers.

Let S be a structure of signature σ. Intentionally, a state of every agent is characterized by a value from S. For the file system example described above the set of elements of S is

$$\texttt{FileId} \times \texttt{FileAttr} \times \texttt{String} \times \texttt{FileId} \times \texttt{Set of FileId} \times \texttt{Boolean}$$

That is we just take the cartesian product of sets representing types of the class fields. The last element in the product stands for the universe of Boolean values $\{$`true, false`$\}$. We added it because, instead of a variable set of agents, it is convenient to think about a fixed infinite set of agents where the additional Boolean valued field `Active` indicates whether the record corresponds to a currently existing element or not. The creation of a new agent corresponds to updating the `Active` field to `true` for a previously inactive agent. Similarly, when the object is destroyed we just flip the value of the `Active` field to `false`.

Let f be a fresh unary functional symbol. The state of the whole system is characterized by a mapping

$$f : I \to S.$$

Namely, agents are identified by elements of I, the state of an agent a is characterized by the value $f(a)$.

2.2 The transition relation

In what follows Diff_k stands for a first order formula asserting that the values of the k variables are different. For example

$$\mathit{Diff}_3(x,y,z) \rightleftharpoons (x \neq y) \wedge (y \neq z) \wedge (x \neq z).$$

Such formulas are expressible in any first-order theory with equality.

In general, the program for a non-deterministic transition τ has the following form:

```
procedure τ
    choose i_1,...,i_k in I, p_1,...,p_m in S
           where Diff_k(i_1,...,i_k) and δ(f(i_1),...,f(i_k),p_1,...,p_m)
        f(i_1) := t_1(f(i_1),...,f(i_k),p_1,...,p_m)
        ...
        f(i_k) := t_k(f(i_1),...,f(i_k),p_1,...,p_m)
```

where, δ is a first-order formula over σ, and $t_1, \ldots, t_k$ are terms over σ.

Thus one step of the system goes like that: k different agents are chosen randomly that satisfy the condition formalized by formula δ. Then, states of the chosen agents are updated accordingly to the program of τ.

A *computation* is a sequence of states (interpretations of f) such that each subsequent state is the result of the transition applied to the previous state for a particular choice of the parameters.

2.3 Properties of the computations

In this paper we are interested in the properties of the following form:

$$\varphi_1 \to \circ_\tau \varphi_2.$$

Here $\circ_\tau$ denotes the well known temporal operator "valid in the next state after transition τ" never mind what choices are made by τ; formulas φ_1, φ_2 are of the following form:

$$\begin{aligned}
\varphi_1 &\rightleftharpoons \forall j_1 \ldots j_s \in I\ (\mathit{Diff}_s(j_1,\ldots,j_s) \to \psi_1(f(j_1),\ldots,f(j_s))),\\
\varphi_2 &\rightleftharpoons \forall j_1 \ldots j_t \in I\ (\mathit{Diff}_t(j_1,\ldots,j_t) \to \psi_2(f(j_1),\ldots,f(j_t))),
\end{aligned}$$

where ψ_1, ψ_2 are first-order formulas over σ with no free variables.

3 The main result

Theorem 1 *For any formulas φ_1, φ_2 and transition τ as described above the relation $\varphi_1 \to \circ_\tau \varphi_2$ is expressible in the first order theory of S.*

Proof. First of all, we expand $\varphi_1 \to \circ_\tau \varphi_2$ according to the definition of $\circ_\tau$. This gives us the following formula:

$$\begin{array}{l}\forall j_1 \dots j_s \in I\ (\mathit{Diff}_s(j_1 \dots j_s) \to \psi_1(f(j_1), \dots, f(j_s))) \to \\ \quad \forall j_1 \dots j_t \in I\ (\mathit{Diff}_t(j_1 \dots j_t) \to \psi_2(f'_\tau(j_1), \dots, f'_\tau(j_t))).\end{array}$$

Here f'_τ stands for the version of f updated according to the transition τ.

Then we expand the definition of τ:

$$\begin{array}{l}\forall j_1 \dots j_s \in I\ [\mathit{Diff}_s(j_1 \dots j_s) \to \psi_1(f(j_1), \dots, f(j_s))] \to \\ \quad \forall j_1 \dots j_t \in I\ [\mathit{Diff}_t(j_1 \dots j_t) \to \\ \quad\quad \forall i_1 \dots i_k \in I\ \forall p_1 \dots p_m \in S\ [\mathit{Diff}_k(i_1, \dots, i_k) \to \\ \quad\quad\quad \delta(f(i_1), \dots, f(i_k), p_1, \dots, p_m) \to \\ \quad\quad\quad\quad \psi_2(f''(j_1, \boldsymbol{i}, \boldsymbol{p}), \dots, f''(j_t, \boldsymbol{i}, \boldsymbol{p}))]],\end{array}$$

where $\boldsymbol{i}$, $\boldsymbol{p}$ are abbreviations for $i_1, \dots, i_k$, and $p_1, \dots, p_m$ correspondingly, and f'' is defined in the following way:

$$f''(j, \boldsymbol{i}, \boldsymbol{p}) = \begin{cases} t_l(f(i_1), \dots, f(i_k), p_1, \dots, p_m), & \text{if } j = i_l, 1 \le l \le k; \\ f(j), & \text{otherwise.} \end{cases}$$

Lemma 1 *The right hand side of the implication $\varphi_1 \to \circ_\tau \varphi_2$ is equivalent to a conjunction of formulas of the form*

$$\forall j_1 \dots j_n \in I[\mathit{Diff}_n(j_1, \dots, j_n) \to \beta(f(j_1), \dots, f(j_n))]$$

where $\beta(x_1, \dots, x_n)$ is a first-order formula over σ.

Proof. Begin by moving all the universal quantifiers out of the formula. The formula in question acquires the form $\forall \boldsymbol{j} \chi(\boldsymbol{j})$ where $\chi(\boldsymbol{j})$ is a boolean combination of equalities $j_p = j_q$ and first-order formulas over σ with substituted terms $f(j)$.

To transform this kind of formula to the desired form we apply the following standard procedure.

First, we consider the following tautology: the complete disjunctive normal form where the equalities $j_p = j_q$ play the roles of propositional variables. Every consistent disjunct $\mathit{Config}_l(\boldsymbol{j})$ represents a pattern of equality over the variables j_p.

Then, instead of the formula $\chi(\boldsymbol{j})$, we consider the following implication (which is equivalent to $\chi(\boldsymbol{j})$ because the antecedent is a tautology):

$$[\bigvee_l \mathit{Config}_l(\boldsymbol{j})] \to \chi(\boldsymbol{j}).$$

This is equivalent to the following conjunction:

$$\bigwedge_l [\mathit{Config}_l(\boldsymbol{j}) \to \chi(\boldsymbol{j})].$$

To complete the proof of the lemma we move the universal quantifier over $\boldsymbol{j}$ inside the conjunction, and then we eliminate all the positive occurrences of equality in Config_l. For example:

$$\forall j_1 j_2 j_3 j_4 (j_1 \neq j_2 \wedge j_2 = j_3 \wedge j_1 = j_4 \rightarrow \chi(j_1, j_2, j_3, j_4))$$

is equivalent to

$$\forall j_1 j_2 (j_1 \neq j_1 \rightarrow \chi(j_1, j_2, j_2, j_1)).$$

Q. E. D.

Lemma 2 *Let $\alpha(x_1, \ldots, x_k)$ and $\beta(y_1, \ldots, y_n)$ be two first order formulas in the signature σ, where all the free variables of the formulas are explicitly shown. Then the following property*

$$\begin{array}{l} \textit{for any function } f : I \rightarrow S \textit{ the following holds:} \\ \forall i_1 \ldots i_k \in I[\mathit{Diff}_k(i_1 \ldots i_k) \rightarrow \alpha(f(i_1), \ldots, f(i_k))] \rightarrow \\ \quad \forall j_1 \ldots j_n \in I[\mathit{Diff}_n(j_1, \ldots, j_n) \rightarrow \beta(f(j_1), \ldots, f(j_n))]. \end{array}$$

is expressible by a first-order formula over σ.

Proof. The proof follows from the following equivalent transformations.

1. For better readability we replace the text in the first line of the property with the second-order quantifier over f, and then move outside the universal quantifier over $\boldsymbol{j}$. As the result we get:

$$\begin{array}{l} \forall f : I \rightarrow S,\ \forall j_1 \ldots j_n \in I[\mathit{Diff}_n(j_1, \ldots, j_n) \rightarrow \\ \quad [\forall i_1 \ldots i_k \in I(\mathit{Diff}_k(i_1 \ldots i_k) \rightarrow \alpha(f(i_1), \ldots, f(i_k))) \rightarrow \\ \qquad \beta(f(j_1), \ldots, f(j_n))]]. \end{array}$$

2. Observe now, that the property starts with two universal quantifiers: we choose any function f, and then any n different values of the function arguments. The rest of the formula is a property about values of the function on these arguments. One can easily see that nothing is lost if we just fix values of $j_1, \ldots, j_n$, e.g. $j_1 = 1$, $j_2 = 2, \ldots, j_n = n$.

Indeed, if the property is true for all values of $\boldsymbol{j}$ then it is certainly true for these particular values. On the other hand, if it is refuted for some particular choice of f and $\boldsymbol{j}$, then one can easily construct f' by permuting some values of f in such a way that the property is refuted for f' and the fixed values of $\boldsymbol{j}$. So, we can transform to the following:

$$\begin{array}{l} \forall f : I \rightarrow S \\ \quad [\forall i_1 \ldots i_k \in I(\mathit{Diff}_k(i_1 \ldots i_k) \rightarrow \alpha(f(i_1), \ldots, f(i_k))) \rightarrow \\ \qquad \beta(f(1), \ldots, f(n))]. \end{array}$$

3. Note now that since f is arbitrary, values $f(1), \dots, f(n)$ are just any values from S. So we get:

$$\begin{array}{l} \forall a_1 \dots a_n \in S, \forall f : I \to S \\ \quad [f(1) = a_1 \wedge \dots \wedge f(n) = a_n \to \\ \quad\quad [\forall i_1 \dots i_k \in I(\mathit{Diff}_k(i_1 \dots i_k) \to \alpha(f(i_1), \dots, f(i_k))) \to \\ \quad\quad\quad \beta(a_1, \dots, a_n)]]. \end{array}$$

or in disjunctive form:

$$\begin{array}{l} \forall a_1 \dots a_n \in S, \forall f : I \to S \\ \quad [\neg(f(1) = a_1 \wedge \dots \wedge f(n) = a_n) \vee \\ \quad\quad \exists i_1 \dots i_k \in I(\mathit{Diff}_k(i_1 \dots i_k) \wedge \neg\alpha(f(i_1), \dots, f(i_k))) \vee \\ \quad\quad \beta(a_1, \dots, a_n)]]. \end{array}$$

4. Since the last line has no occurrences of f, this is equivalent to:

$$\begin{array}{l} \forall a_1 \dots a_n \in S[\beta(a_1, \dots, a_n) \vee \\ \quad \forall f : I \to S[(f(1) = a_1 \wedge \dots \wedge f(n) = a_n) \to \\ \quad\quad \exists i_1 \dots i_k \in I(\mathit{Diff}_k(i_1 \dots i_k) \wedge \neg\alpha(f(i_1), \dots, f(i_k)))]] \end{array}$$

5. Without loss of generality one can assume that values of $i_1 \dots i_k$ in the last line of the formula are restricted with $k + n$. Indeed, with this kind of formula we can only distinguish cases when $i_s = 1, \dots, i_s = n$, $i_s > n$, and equalities $i_{s_1} = i_{s_2}$. In the worst case, after applying the corresponding permutation to f we get $i_1 = n + 1, \dots, i_k = n + k$.

As the result we get the following formula:

$$\begin{array}{l} \forall a_1 \dots a_n \forall a_{n+1} \dots a_{n+k} \in S[\beta(a_1, \dots, a_n) \vee \\ \quad \forall f : I \to S[(f(1) = a_1 \wedge \dots \wedge f(n + k) = a_{n+k}) \to \\ \quad\quad \exists i_1 \dots i_k \in 1, \dots, n + k(\mathit{Diff}_k(i_1 \dots i_k) \wedge \neg\alpha(f(i_1), \dots, f(i_k)))]] \end{array}$$

6. Now, the values $f(i)$ for $i > n+k$ could be ignored. So instead of a function we can consider sequences of integers:

$$\begin{array}{l} \forall a_1 \dots a_{n+k} \in S[\beta(a_1, \dots, a_n) \vee \\ \quad \exists i_1 \dots i_k \in \{1, \dots, n + k\}(\mathit{Diff}_k(i_1 \dots i_k) \wedge \neg\alpha(a_{i_1}, \dots, a_{i_k}))] \end{array}$$

7. To finish the proof note that existential quantifier over the finite set $\{1, \dots, n + k\}$ could be replaced with the corresponding finite disjunction.

Q. E. D.

Note that this reduction from the second order language to the first order turns out to be quite simple. It is possible that the result was known, but we don't know any relevant references.

Corollary 1 *Suppose the first order theory of S is decidable, then the relation $\varphi_1 \to \circ_\tau \varphi_2$ is decidable too.*

Proof. Indeed, according to the theorem the relation is expressible by a first order formula over S. So, it is decidable.

4 Conclusion

Let K denote the class of computational systems satisfying the following conditions:

1. The system state is characterized by a finite collections of agents.
2. Each of the agents is characterized by an element of the structure S.
3. A transition of the system consists of the following three steps: some new agents arrive, some previously active agents leave the system, and some other agents are updated. The total number of created, updated and removed agents is uniformly bounded.
4. All the updates are expressible by first order formulas over S.

Corollary 2 *Suppose the first order theory of S is decidable. Then assume-guarantee properties $\varphi_1 \to \circ_\tau \varphi_2$ of systems from the class K are decidable provided the precondition φ_1 and the postcondition φ_2 are expressible by first order formulas over S.*

One example of S is Presburger Arithmetic of addition [5]; see [6] for more.

5 Acknowledgements

The authors are thankful to anonymous referees for numerous helpful remarks to the text.

References

1. Foundations of Software Engineering group, Microsoft Research, http://research.microsoft.com/fse/
2. AsmL: The Abstract State Machine Language. Reference Manual. Modeled Computation LLC, 2002. http://research.microsoft.com/fse/asml/
3. Wolfgang Grieskamp, Yuri Gurevich, Wolfram Schulte, and Margus Veanes. *Generating Finite State Machines from Abstract State Machines.* In ISSTA 2002, International Symposium on Software Testing and Analysis, July 2002.
4. Margus Veanes and Rostislav Yavorsky. *Combined Algorithm for Approximating a Finite State Abstraction of a Large System.* In SCESM 2003, 2-nd International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools, May 2003.
5. George S. Boolos, John P. Burgess, and Richard C. Jeffrey. **Computability and Logic.** Cambridge University Press, 2002.
6. M.O. Rabin. *Decidable theories.* In J. Barwise, editor, Handbook of Mathematical Logic, pp. 595-627, North Holland, 1977.