Polymorphic Systems with Arrays: Decidability and Undecidability* (Extended Abstract of Work in Progress)

Ranko Lazić[1]**, Tom Newcomb[2], and Bill Roscoe[2]

[1] Department of Computer Science, University of Warwick, UK
Ranko.Lazic@dcs.warwick.ac.uk
[2] Computing Laboratory, University of Oxford, UK
Tom.Newcomb@comlab.ox.ac.uk
Bill.Roscoe@comlab.ox.ac.uk

Abstract. Polymorphic systems with arrays (PSAs) is a general class of nondeterministic reactive systems. A PSA is polymorphic in the sense that it depends on a signature, which consists of a number of type variables, and a number of symbols whose types can be built from the type variables. Some of the state variables of a PSA can be arrays, which are functions from one type to another. We present several new decidability and undecidability results for the parameterised control-state reachability problem on subclasses of PSAs.

1 Introduction

Model checking (e.g. [1]) is the approach to verification where it is checked algorithmically whether a model of a given system satisfies a correctness property. Compared with testing and theorem proving, which are the other main techniques for verification, model checking is distinguished by being exhaustive, automatic, and returning counter-examples when the model does not satisfy the property. Due to these features, model checking is being used in industry.

In its basic form, model checking applies to finite-state systems, where correctness is either satisfaction of a temporal logic formula, or refinement of another finite-state system. Such verification problems are decidable, and efficient algorithms for solving them have been developed.

Model checking of infinite-state systems has also been a successful research area (e.g. [7]). Without the finite-state restriction, many verification problems are undecidable, such as the Halting Problem for Turing machines. However, for many interesting classes of infinite-state systems and correctness properties, verification is decidable. Research has concentrated on classifying verification problems as undecidable or decidable, and

* We acknowledge support from the EPSRC Standard Research Grant 'Exploiting Data Independence', GR/M32900. The first author was also supported by a research grant from the Intel Corporation, the second author by QinetiQ Malvern, and the third author by the US ONR.

** Also affiliated to the Mathematical Institute, Serbian Academy of Sciences and Arts, Belgrade.

Proceedings of the 1st South-East European Workshop on Formal Methods, SEEFM'03
November 2003, Thessaloniki, Greece

in the latter case determining their complexity. Much valuable work has also been done on semi-algorithms, abstraction, combinations of model checking, theorem proving and testing, etc.

In this paper, we introduce polymorphic systems with arrays (PSAs). A PSA is polymorphic in the sense that it depends on a signature, which consists of a number of type variables, and a number of symbols whose types can be built from the type variables. Some of the state variables of a PSA can be arrays, which are functions from one type to another.

A signature is instantiated by assigning sets to its type variables, and concrete elements or operations to its symbols. Given a PSA and an instantiation of its signature, we get a model which is a transition system, and which is finite-state if all the type variables are assigned finite sets.

We study parameterised verification of PSAs, so a PSA will in fact be equipped with a set of instantiations of its signature. Its model will consist of all transition systems for the given instantiations. Normally, infinitely many instantiations are given, resulting in an infinite-state model.

PSAs generalise data-independent systems with arrays [10, 8, 14, 12]. They are PSAs whose signatures consist of type variables and binary predicate symbols on some of those type variables, which are always instantiated as equality predicates. In other words, data-independent systems with arrays are polymorphic, but can use at most equality on elements of variable types.

Another interesting subclass of PSAs which we consider in this paper generalises data-independent systems with arrays by allowing some type variables to be equipped with linear-order predicates.[1]

The definition of PSAs allows a general form of assignment to array variables, whereas the research on data-independent systems with arrays considered specific operations, such as a write to an array component, or a reset of a whole array to a value. This corresponds to studying subclasses of PSAs where only certain kinds of assignment to array variables are allowed.

There is strong practical motivation to study PSAs, as they arise wherever a system is parameterised by data types, and contains arrays. Examples include: database with record locking and two security levels of records, which is parameterised by types of record indices and record contents [10]; fault-tolerant memory, parameterised by types of memory addresses and storable data [8]; cache-coherence protocols, parameterised by types of processor indices, memory addresses and storable data (e.g. [2, 13]).

PSAs are related to a number of other classes of systems: a marking of a Petri net can be seen as an array which maps each token to the place which contains it [14]; a configuration of a Broadcast Protocol [3] can be regarded as an array mapping process indices to process states; a system with sets can be seen as a system with arrays by representing a set by an array containing booleans. These relationships, together with the fact that they are not equivalences, strengthen the case for studying PSAs and their verification.

Another view of PSAs worth mentioning is that, for example, a PSA whose signature consists of a linearly-ordered type variable X, and which has an array variable indexed

[1] A linear-order predicate can express the equality predicate by $a = b \Leftrightarrow a \leq b \wedge b \leq a$.

by X and a number of variables of type X, can be seen as a Turing machine which is parameterised by the size of its tape, has a number of heads, and is only able to compare the positions of two heads (as well as move a head to a nondeterministic new position), but not move a head left or right by exactly one position.

In this paper, we present a few new undecidability results for parameterised verification of subclasses of PSAs, and one new decidability result. Both kinds of results are useful for guiding work on verification tools.

Proofs will be available in the full paper [9].

2 Polymorphic systems with arrays

To define PSAs, we start with the syntax of types.[2] For simplicity, we have basic types built from type variables, products and non-empty sums (i.e. disjoint unions), and function types from one basic type to another. Function types will be used as types of array variables, and also as types of signature symbols such as equality predicates.

$$B ::= X \mid B_1 \times \cdots \times B_n \mid B_1 + \cdots + B_{n \geq 1}$$
$$T ::= B \mid B \to B'$$

Next we need a syntax of terms, which will be used to form one-step computations of PSAs. The terms are built from term variables, tuple formation, tuple projection, sum injection, sum case, and function application. For simplicity, instead of having a λ-abstraction term construct for forming functions, PSAs will have a general form of assignment to array variables.

$$\begin{aligned} t ::= {} & x \mid (t_1, \ldots, t_n) \mid \pi_i(t) \mid \\ & \iota_i^B(t) \mid \mathit{case}\ t\ \mathit{of}\ x_1.t'_1\ \mathit{or}\ \ldots\ \mathit{or}\ x_n.t'_n \mid \\ & t_1\, t_2 \end{aligned}$$

We consider only well-typed terms. A signature consists of a finite set Ω of type variables, and a type context Γ which is a finite sequence $x_1 : T_1, \ldots, x_n : T_n$ of typed and mutually distinct term variables, where the types T_i can contain only type variables from Ω. A well-typed term-in-context is written $\Omega, \Gamma \vdash t : T$, where these valid type judgements are deduced by standard typing rules.

Using the types and terms above, we can for example express:

- the singleton type *Unit* as the empty product, and its unique element as the empty tuple;
- the boolean type *Bool* as the sum of two *Unit* types, and terms *false*, *true*, and *if* t *then* t'_1 *else* t'_2;
- for any positive n, the n-element enumerated type Enum_n as the sum of n *Unit* types, its elements e_1, ..., e_n, and a case term.

[2] For computations within PSAs, we use a fragment of a typed λ-calculus (e.g. [11]).

We can also express any given operation on the *Bool* and $Enum_n$ types, of any arity.

Semantics of types is defined as follows. A finite set Ω of type variables is instantiated by a mapping ω to non-empty sets. For any type T such that $Vars(T) \subseteq \Omega$, its semantics with respect to ω is a non-empty set $\mathrm{J}T\mathrm{K}_\omega$, which is defined in the usual way.

For semantics of terms, a signature (Ω, Γ) is instantiated by a ω as above, and a mapping $\gamma \in \mathrm{J}\Gamma\mathrm{K}_\omega$, i.e. $Dom(\gamma) = Dom(\Gamma)$ and $\gamma\mathrm{J}x\mathrm{K} \in \mathrm{J}T\mathrm{K}_\omega$ for all $x : T$ in Γ. For any well-typed term-in-context $\Omega, \Gamma \vdash t : T$, its semantics with respect to (ω, γ) is an element $\mathrm{J}t\mathrm{K}_{\omega,\gamma}$ of $\mathrm{J}T\mathrm{K}_\omega$, and is defined in the standard way.

Definition 1. *A PSA is a 5-tuple $(\Omega, \Gamma, \Theta, R, I)$ such that:*

- *(Ω, Γ) is a signature, consisting of type variables and typed term variables (i.e. typed constant or operation symbols) which the PSA is parameterised by.*
- *Θ is a type context disjoint from Γ, and such that $(\Omega, \Gamma\Theta)$ is a signature. Θ specifies the state variables of the PSA and their types. According to its type, a state variable is either basic or an array.*
- *R is a finite set of instructions. Each $J \in R$ is a set of assignments (performed simultaneously) to mutually distinct state variables. The form of each assignment is:*
  - *For basic state variables $b : B$, it is $b := ?\Phi : d \cdot t$, where Φ is a type context disjoint from $\Gamma\Theta$ and containing only basic types, and*

$$\Omega, \Gamma\Theta\Phi \vdash d : Bool$$
$$\Omega, \Gamma\Theta\Phi \vdash t : B$$

    The semantics will be that Φ consists of parameters whose values are chosen nondeterministically subject to satisfying d, and then the value of t is assigned to b.
  - *For array state variables $a : B \to B'$, it is $a[x : c] := ?\Phi : d \cdot t$, where $\Gamma\Theta$, $\langle x : B\rangle$ and Φ are mutually disjoint type contexts and Φ contains only basic types, and*

$$\Omega, \Gamma\Theta\langle x : B\rangle \vdash c : Bool$$
$$\Omega, \Gamma\Theta\langle x : B\rangle\Phi \vdash d : Bool$$
$$\Omega, \Gamma\Theta\langle x : B\rangle\Phi \vdash t : B'$$

    The semantics will be that, for all values of x such that c is satisfied, values for the parameters in Φ are chosen nondeterministically (and possibly differently for different values of x) subject to satisfying d, and then the value of t is assigned to $a[x]$.
- *I is a set of instantiations of (Ω, Γ).*

The following are some special cases of array assignment:

Write. Assigning t' to $a[t]$:

$$a[x : x = t] := ?\langle\rangle : true \cdot t'$$

Reset. Assigning t' to all components of a:

$$a[x : true] := ?\langle\rangle : true \cdot t'$$

Assign. Assigning an array a' to a:

$$a[x : true] := ?\langle\rangle : true \cdot a'x$$

Choose. Nondeterministically choosing a whole array:

$$a[x : true] := ?\langle y : B'\rangle : true \cdot y$$

Map. Applying an operation componentwise to several arrays:

$$a[x : true] := ?\langle\rangle : true \cdot f(a'_1 x, \ldots, a'_n x)$$

Cross-section. For example, a row t of an array $a : (B_1 \times B_2) \to B'$ can be assigned to using the condition $\pi_1(x) = t$.

Definition 2. *The semantics of a PSA $(\Omega, \Gamma, \Theta, R, I)$ is the transition system $(S, \to)$ defined as follows:*

- *The set of states S consists of all (ω, γ, θ) such that $(\omega, \gamma) \in I$ and $\theta \in \mathrm{J}\Theta\mathrm{K}_\omega$.*
- *$(\omega, \gamma, \theta) \to (\omega', \gamma', \theta')$ iff $\omega' = \omega$, $\gamma' = \gamma$, and there exists $J \in R$ which can produce θ' from θ. This means that, for each assignment $b := ?\Phi : d \cdot t$ in J, there exists $\phi \in \mathrm{J}\Phi\mathrm{K}_\omega$ such that*

$$\begin{aligned} \mathrm{J}d\mathrm{K}_{\omega,\gamma\theta\phi} &= tt \\ \theta'\mathrm{J}b\mathrm{K} &= \mathrm{J}t\mathrm{K}_{\omega,\gamma\theta\phi} \end{aligned}$$

Also, for each assignment $a[x : c] := ?\Phi : d \cdot t$ in J, where $\Theta(a) = B \to B'$, and for each $v \in \mathrm{J}B\mathrm{K}_\omega$ such that $\mathrm{J}c\mathrm{K}_{\omega,\gamma\theta\{x\mapsto v\}} = tt$, there exists $\phi_v \in \mathrm{J}\Phi\mathrm{K}_\omega$ such that

$$\begin{aligned} \mathrm{J}d\mathrm{K}_{\omega,\gamma\theta\{x\mapsto v\}\phi} &= tt \\ \theta'\mathrm{J}a\mathrm{K}(v) &= \mathrm{J}t\mathrm{K}_{\omega,\gamma\theta\{x\mapsto v\}\phi} \end{aligned}$$

Finally, for each $v \in \mathrm{J}B\mathrm{K}_\omega$ such that $\mathrm{J}c\mathrm{K}_{\omega,\gamma\theta\{x\mapsto v\}} = ff$, we have $\theta'\mathrm{J}a\mathrm{K}(v) = \theta\mathrm{J}a\mathrm{K}(v)$.

3 Undecidability results

We consider the following classes of PSAs:

$X \times X$-to-Bool. This class consists of all PSAs $(\Omega, \Gamma, \Theta, R, I)$ such that:
- $\Omega = \{X\}$ and $\Gamma = \langle =_X \colon X \times X \to Bool\rangle$;
- there is only one array variable in Θ, and it is of type $X \times X \to Bool$;
- the array assignments in R are only writes and resets;

- I consists of all (ω, γ) such that ω assigns to X a set of the form $\hat{k} = \{1, \ldots, k\}$, and γ assigns to $=_X$ the equality predicate on $\hat{k}$.

$X \times Y$-to-Bool. Here X and Y are distinct type variables, and the restrictions are:
- $\Omega = \{X, Y\}$ and $\Gamma = \langle =_X \colon X \times X \to \mathit{Bool}, =_Y \colon Y \times Y \to \mathit{Bool} \rangle$;
- there is only one array variable in Θ, and it is of type $X \times Y \to \mathit{Bool}$;
- the array assignments in R are only writes and resets;
- I consists of all (ω, γ) such that ω assigns to X and Y some $\hat{k}$ and $\hat{l}$, and γ assigns to $=_X$ and $=_Y$ the equality predicates.

X-to-Y,Z. Here X, Y, Z are distinct type variables, and the restrictions are:
- $\Omega = \{X, Y, Z\}$ and $\Gamma = \langle =_X \colon X \times X \to \mathit{Bool}, =_Y \colon Y \times Y \to \mathit{Bool}, =_Z \colon Z \times Z \to \mathit{Bool} \rangle$;
- there are only two array variables in Θ, and they are of types $X \to Y$ and $X \to Z$;
- the array assignments in R are only writes and resets;
- I consists of all (ω, γ) such that ω assigns to X, Y, Z some $\hat{k}$, $\hat{l}$, $\hat{m}$, and γ assigns to $=_X$, $=_Y$, $=_Z$ the equality predicates.

X,$\leq$-to-Y. Here X and Y are distinct type variables, and the restrictions are:
- $\Omega = \{X, Y\}$ and $\Gamma = \langle \leq_X \colon X \times X \to \mathit{Bool}, =_Y \colon Y \times Y \to \mathit{Bool} \rangle$;
- there is only one array variable in Θ, and it is of type $X \to Y$;
- the array assignments in R are only writes and resets;
- I consists of all (ω, γ) such that ω assigns to X and Y some $\hat{k}$ and $\hat{l}$, $\gamma \mathrm{J} \leq_X \mathrm{K}$ is the ordering on $\hat{k}$, and $\gamma \mathrm{J} =_Y \mathrm{K}$ is the equality predicate on $\hat{l}$.

Given a PSA, a state variable b of type Enum_n[3], and $i, j \in \{1, \ldots, n\}$, the *control-state reachability problem* is to decide whether there exists a sequence of transitions from a state (ω, γ, θ) with $\theta \mathrm{J} b \mathrm{K} = i$ to a state $(\omega, \gamma, \theta')$ with $\theta' \mathrm{J} b \mathrm{K} = j$. The problem of checking any safety property can be reduced to this problem.

Theorem 1. *The control-state reachability problem is undecidable for each of the classes $X \times X$-to-Bool, $X \times Y$-to-Bool, X-to-Y,Z, and X,$\leq$-to-Y.*

Proof. In each case, the proof is by a reduction from the reachability problem for 2-counter automata.

In [14], it was shown that control-state reachability is decidable for systems with an array from X with equality to an enumerated type. In [12, Chapter 8], decidability of the same problem was shown for systems with an array from X with equality to Y with equality. Theorem 1 tells us that decidability fails when the former arrays are generalised to two-dimensional, and when the latter arrays are generalised to X with a linear ordering.

By regarding X as the type of processor indices, Y as the type of memory addresses, and *Bool* as the type of storable data, the class $X \times Y$-to-Bool contains classes of cache-coherence protocols (e.g. [2, 13]). By Theorem 1, any decidability result for control-state reachability for such a class of protocols must depend on some properties of the protocols which are not common to the whole class $X \times Y$-to-Bool.

[3] Any tuple of variables whose types do not contain type variables is isomorphic to a variable of type Enum_n.

It was also shown in [14] that control-state reachability is undecidable for systems with two arrays from X with equality to Y with equality. Theorem 1 states that undecidability remains for the less expressive class X-to-Y,Z.[4]

4 Decidability result

Let X,$\leq$-to-Enum be the class of all PSAs $(\Omega, \Gamma, \Theta, R, I)$ such that:

- $\Omega = \{X\}$ and $\Gamma = \langle \leq_X : X \times X \to \mathit{Bool} \rangle$;
- the type of any array variable in Θ is of the form $X \to \mathit{Enum}_m$;
- in every array assignment $a[x : c] := ?\Phi : d \cdot t$ in R, d does not contain x;
- I consists of all (ω, γ) such that ω assigns to X some $\hat{k}$, and γ assigns to $\leq_X$ the linear ordering on $\hat{k}$.

Theorem 2. *The control-state reachability problem is decidable for the class X,$\leq$-to-Enum.*

Proof. The proof uses the set-saturation methods in [5].

Theorem 2 strengthens the decidability result in [14] for systems with arrays from X with equality to enumerated types.

The restriction on array assignments is necessary because a counter can be represented by the number of indices of an array at which a certain value $i \in \{1, \ldots, m\}$ is stored. An array assignment

$$a[x : \mathit{true}] := ?\langle\rangle : a\, x \neq e_i \cdot a\, x$$

can then be executed iff the counter is zero, enabling a reduction of the reachability problem for 2-counter automata.

Example 1. The class X,$\leq$-to-Enum contains a model of the Bully Algorithm for leadership election in a distributed system in which process identifiers are linearly ordered [6]. The parameterisation by $(X, \leq)$ captures the parameterisation by the number of processes. By Theorem 2 and by reducing safety properties to control-state reachability, many interesting properties can be decided, such as:

- there are never two distinct coordinators;
- a process cannot receive a coordinator announcement from a process whose identifier is smaller than the identifier of the previous coordinator without detecting that the previous coordinator had failed;
- there is never a coordinator and a process with a greater identifier which believes it has a coordinator.

[4] This class is less expressive because the types prevent the values contained in the two arrays to be mixed.

5 Future work

Related to Theorem 1, undecidability of control-state reachability for classes of PSAs which contain the reset operation on arrays can be used to deduce undecidability of checking certain temporal properties for classes of PSAs which do not contain resets.

The part of Theorem 1 for the class $X \times Y$-to-Bool implies undecidability of model checking safety properties for a class of systems obtained by generalising Broadcast Protocols [3] to rectangular networks, where broadcasts restricted to a row or a column are possible.

A corollary of Theorem 2 is a generalisation of the decidability result for model checking safety properties of Broadcast Protocols [4], where linear-ordering comparisons of process indices are allowed.

Other on-going work includes generalising the decidability results in [14] and [12, Chapter 8], and Theorem 2 to classes of PSAs with more than one array.

Acknowledgements

We thank Sara Kalvala for a useful discussion.

References

1. E.M. Clarke, O. Grumberg and D.A. Peled, *Model Checking*, MIT Press, January 2000.
2. G. Delzanno, *Automatic verification of parameterised cache coherence protocols*, Proceedings of the 12th International Conference on Computer-Aided Verification (CAV 2000), Lecture Notes in Computer Science 1855, 53–68, Springer, July 2000.
3. E.A. Emerson and K.S. Namjoshi, *On model checking for non-deterministic infinite-state systems*, Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS), 1998.
4. J. Esparza, A. Finkel and R. Mayr, *On the verification of broadcast protocols*, Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS), 352–359, July 1999.
5. A. Finkel and Ph. Schnoebelen, *Well-structured transition systems everywhere!*, Theoretical Computer Science 256 (1–2): 63–92, 2001.
6. H. Garcia-Molina, *Elections in a distributed computing system*, IEEE Transactions on Computers 31 (1): 48–59, 1982.
7. International Workshops on Verification of Infinite-State Systems. `http://www.lsv.ens-cachan.fr/infinity03/`
8. R.S. Lazić, T.C. Newcomb and A.W. Roscoe, *On model checking data-independent systems with arrays without reset*, Programming Research Group Research Report RR-02-02, 31 pages, Oxford University Computing Laboratory, January 2002. Revised version to appear in the journal Theory and Practice of Logic Programming (TPLP), Cambridge University Press.
9. R.S. Lazić, T.C. Newcomb and A.W. Roscoe, *Polymorphic systems with arrays: decidability and undecidability*, Research Report, Department of Computer Science, University of Warwick, in preparation.

10. R. Lazić and A.W. Roscoe, *Verifying determinism of concurrent systems which use unbounded arrays*, Proceedings of the 3rd International Workshop on Verification of Infinite State Systems (INFINITY '98), Report TUM-I9825, 2–8, Technical University of Munich, July 1998.
11. J.C. Mitchell, *Type Systems for Programming Languages*, in [15], 365–458.
12. T.C. Newcomb, *Model Checking Data-Independent Systems with Arrays*, D.Phil. thesis draft, Computing Laboratory, Oxford University, June 2003.
13. S. Qadeer, *Verifying sequential consistency on shared-memory multiprocessors by model checking*, Research Report 176, Compaq, Palo Alto, 2001.
14. A.W. Roscoe and R.S. Lazić, *What can you decide about resetable arrays?*, Proceedings of the 2nd International Workshop on Verification and Computational Logic (VCL'2001), Technical Report DSSE-TR-2001-3, 5–23, Declarative Systems and Software Engineering Research Group, Department of Electronics and Computer Science, University of Southampton, September 2001.
15. J. van Leeuwen (editor), *Formal Models and Semantics*, Handbook of Theoretical Computer Science, volume B, Elsevier, 1990.