Verifying Determinism of Concurrent Systems Which Use Unbounded Arrays (Extended Abstract)

Ranko Lazić*† Bill Roscoe‡

Submitted to INFINITY '98

Abstract

Our main result says that determinism of a concurrent system which uses unbounded arrays (i.e. memories) can be verified by considering an appropriate finite array size.

That is made possible by restricting the ways in which array indices and values can be used within the system. The restrictions are those of data independence: the system must not perform any operations on the indices and values, but it is only allowed to input them, store them, and output them. Equality tests between indices are also allowed.

The restrictions are satisfied by many concurrent systems which use arrays to model memories or databases. As a case study, we have verified that a database system which allows users to lock, read and write records at multiple security levels is secure.

1 The Parameterised Verification Problem

Concurrent systems are frequently infinite-state because they have parameters that can vary unboundedly. For example, a memory cache is likely to be parameterised by the data types of addresses and values, a protocol for fault-tolerance is likely to be parameterised by the number of nodes in the network, etc. Given such a system, we typically want to be able to consider all possible instantiations of its parameters, rather than having to restrict our attention to one instantiation at a time. Thus we come to the Parameterised Verification Problem:

PVP. Given a parameterised concurrent system `P` and a condition `C`, does `P` satisfy `C` for all instantiations of the parameters? ■

There has recently been much research on the PVP. It is undecidable in general [AK86], and so most effort has been put into either finding restricted versions of it which are decidable (see e.g. [Wol86, JP93, ID96a, HB95, HDB97, GS92, YJL96, EN96]), or providing procedures whose termination is unpredictable or which require user assistance (see e.g. [HL95, HGD95, CZ+97, HIB97, CGJ95, ID96b, KM+97]).

The main result of this paper, Theorem 1, says that in order to verify that a concurrent system `P` which uses unbounded arrays (i.e. memories) is deterministic for all array sizes (finite or infinite), it suffices to consider a single appropriate finite size. In other words, the theorem provides a decision procedure for the following restricted version of PVP: the parameters are data types of array indices and values, and the condition `C` is the property of being deterministic. The procedure is guaranteed to terminate whenever the data types sizes which it suffices to consider can be computed in a finite time and the resulting instantiation of `P` is finite-state (which are both issues that are orthogonal to the parameterisation).

*Oxford University Computing Laboratory, U.K. and Mathematical Institute, Belgrade University, Yugoslavia. Supported by a Junior Research Fellowship at Christ Church (Oxford), and previously by a Domus and Harmsworth Senior Scholarship at Merton College (Oxford) and by a scholarship from Hajrija & Boris Vukobrat and Copechim France S.A.

†The contact author. Address: Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, U.K. Tel: +44 1865 273 838. Fax: +44 1865 273 839. Email: rsl@comlab.ox.ac.uk.

‡Oxford University Computing Laboratory, U.K. Supported by grants from the U.S. Office of Naval Research.

Since checking whether the appropriate instantiation of `P` is deterministic can be done by a single run of a model checker,[1] the paper contributes to turning model checkers into practical tools for *verification*, rather than primarily *refutation* [Rus97].

Before we present Theorem 1 in Section 4, we shall explain what we mean by arrays whose index and value types are parameters, and review determinism, a condition with very important applications in the field of computer security. In the remaining two sections, we shall outline the database system case study, and point towards future work.

In the space available, we could not afford to go into technical details. The technical report [LR98a] contains the full paper, including a sketch of the proof of Theorem 1.

2 Unbounded Arrays

The main novelty in the paper is that we consider concurrent systems which can use arrays whose indices and values come from variable types, i.e. data types that are parameters of the system. Such an array is unbounded because its size is the size of the type of its indices, and we allow type parameters to vary unboundedly.

Our main result, Theorem 1, is made possible by restricting the ways in which array indices and values can be used within the system. More precisely, the system is required to be *data independent* with respect to the types of indices and values: it must not apply any operations to elements of those types, but it is only allowed to input them, store them, and output them. Equality tests between indices are also allowed. From another point of view, we have shown that it is possible to relax the data independence conditions by allowing arrays and still obtain finite instantiations which suffice for verification.

Many systems which use arrays to model memories or databases satisfy the data independence conditions on indices and values. A typical example is a memory cache which uses an array to model the main memory that it is interacting with, and whose replacement policy does not involve calculation. Indeed, the only operation such a cache needs to perform on indices or values is equality tests between indices.

Theorem 1 is also made possible by having the indices and values belong to *different* data types, so that for example nested indexing is not allowed.

For simplicity, we shall be considering a single variable type `X` of indices and a single variable type `Y` of values. Let `Array(X, Y)` denote the type of all arrays with indices from `X` and values from `Y`. For creating and using arrays, we allow only the following operations, whose meanings are self-explanatory:

```
init   :: Y -> Array(X, Y)
getval :: (Array(X, Y), X) -> Y
update :: (Array(X, Y), X, Y) -> Array(X, Y)
```

In addition to allowing arrays, Theorem 1 allows constants of variable types, as well as many-valued predicates (i.e. functions into fixed finite types) on variable types which are not types of array values. These two further relaxations of data independence were needed in the database system case study (see Section 5).

3 Determinism

A concurrent system `P` is said to be *deterministic* if, from the point of view of its environment, it cannot exhibit any nondeterminism. More precisely, there must not exist an execution of a sequence t of communications after which `P` can accept some communication a and a possibly different execution of t after which `P` can refuse a. In the process algebra CSP [Hoa85, Ros98], it is appropriate to also require livelock (i.e. divergence) freedom, so that the condition of being deterministic can be formalised as:

- $(t\hat{}\langle a\rangle, \{\}) \in \mathit{failures}_{\perp}(\mathtt{P}) \Rightarrow (t, \{a\}) \notin \mathit{failures}_{\perp}(\mathtt{P})$, and

[1] Determinism checking is supported directly in the model checker FDR [Ros98, FS97]. In other model checkers, some preliminary transformations may be necessary: for example, the condition of being deterministic can be expressed in a linear temporal logic on a product of the labelled transition system of `P` with itself.

- *divergences*(P) = {}.

Determinism has been studied most extensively in CSP. Apart from its great significance in the theory, it has found very important applications in the field of computer security. Namely, it has been established [RWW96, Ros95, Wul97, Ros98] that the key to specifying that there is no information flow across a concurrent system from a high security user to a low security user is the determinism of a suitably formed abstraction: what the system looks like to the low user when the actions of the high user are turned into internal nondeterminism.

4 The Theorem

For concreteness and practical applicability, we have stated and proved Theorem 1 in terms of CSP_M [Sca98, Ros98], the machine-readable version of CSP which is the language used in the model checker FDR [Ros98, FS97].

The following quantities, defined for a concurrent system P and a variable data type X, will be used in the theorem to compute the sufficient size for the type of indices:

$W^{\mathtt{P}}_{\mathtt{X}}$ is the maximum number of elements of type X that P ever has to store for future use. Here it is crucial that we do not need to count elements which are stored only because they are indices of array components that have previously been assigned to or read from. This enables $W^{\mathtt{P}}_{\mathtt{X}}$, and hence the sufficient size for X in the theorem, to be finite even when P may during its execution access an unbounded number of different array components.

$L^{\mathtt{P}}_{?,\mathtt{X}}$ is the maximum number of elements of type X that can be input in a *single* communication of P.

$L^{\mathtt{P}}_{\overline{?},\mathtt{X}}$ is the maximum number of elements of type X that can be input in a *single* communication of P without being recorded in it. (This quantity can be made nonzero only by renamings which omit inputs from communications.)

$L^{\mathtt{P}}_{\sqcap,\mathtt{X}}$ is the maximum number of elements of type X that can be chosen in a *single* internal nondeterministic choice in P.

The quantity $W^{\mathtt{P}}_{\mathtt{X}}$ virtually always dominates the calculations. In fact, in most practical examples, including the case study in the next section, the remaining three quantities have values 1, 0, 0 or 1, 0, 1 respectively.

For simplicity, we state the theorem with only one many-valued predicate and only one constant.

Theorem 1 *Suppose* P *is a concurrent system with two variable types* X *and* Y *with respect to which it is data independent, except that:*

- *it uses arrays of type* `Array(X, Y)`*, i.e. arrays with indices from* X *and values from* Y*,*
- *it uses a K-valued predicate* r *on* X*,*[2] *and*
- *it has a constant* c *of type* Y*.*

Suppose also that P *does not use any equality tests between values from* Y*, and that no state of* P *has two possible communications that differ only in outputs of type* Y*.*

Let $B = 2 \times W^{\mathtt{P}}_{\mathtt{X}} + \max(L^{\mathtt{P}}_{?,\mathtt{X}} + L^{\mathtt{P}}_{\overline{?},\mathtt{X}}, L^{\mathtt{P}}_{\sqcap,\mathtt{X}})$.

Then P *is deterministic for all instantiations of* X*,* Y*,* r *and* c *(with finite or infinite* X *and* Y*), provided it is deterministic for the following instantiation:*

- X *is defined to be any type with exactly* $(B + 2) \times K$ *members,*
- Y *is defined to be any type with exactly* 2 *members, say* `{0, 1}`*,*

[2]When P does not use a predicate, the theorem can be applied with $K = 1$.

- r *is defined to map exactly* $B + 2$ *elements of* X *to each of the* K *elements in its range, and*

- c *is defined to have any fixed value from* Y*, say* 0*.*

Proof. The proof uses symbolic labelled transition systems [HL95, LR98b] and factoring out of symmetry [CE+96, ES96, ID96a, Jen96], and involves much additional development to be able to deal with arrays and with the stated definition of $W_{\mathtt{X}}^{\mathtt{P}}$.

A sketch can be found in [LR98a]; the full proof will appear in [Laz98]. ■

It is not impossible for B to be infinite, in which case the theorem is unlikely to be useful in practice. (As we have said, the question of whether B is infinite is orthogonal to the parameterisation by X and Y.)

4.1 Decision Procedures

The theorem immediately gives us a decision procedure for this restricted version of the PVP: compute B from the description of P, and check whether the resulting instantiation of P is deterministic. As we have remarked, the procedure terminates provided B is computable in a finite time and the instantiation of P is finite-state.

The efficiency of the determinism checking phase can be considerably improved by automatically factoring out the symmetry associated with the type X and by allowing at most one input of type Y to have the value 1 in any execution. Moreover, by performing a suitable *symbolic check*, the computation of B can effectively be done lazily during the check itself, resulting in greater accuracy.

It is also possible to automatically check whether the assumptions of the theorem, or at least their suitable stronger versions, are satisfied.

At the time of writing, extensions to the model checker FDR to fully support the obtained decision procedures are being planned.

4.2 Related Work

We are not aware of any general results in the literature enabling systems that can access unboundedly many different array components (i.e. memory locations) during their execution to be verified by reduction to finite instantiations (as required by most model checkers). Such systems were typically verified by one of the following methods:

- applying existing data independence theorems together with special-case observations or arguments, as in [HM+95, ID96a, LR96, Ros98];

- symbolic execution which is not guaranteed to always terminate, or which relies on the number of array accesses being bounded, as in [VB+97, HIB97];

- special-case abstractions, as in [Gra94].

We also cannot point to previous results facilitating reduction to finite instantiations when many-valued predicates on variable types are present, although it is substantially easier to extend the known methods to accomodate them than it is for unbounded arrays.

5 A Case Study

With the help of Theorem 1, we have verified that a database system which allows users to lock, read and write records at multiple security levels is secure. The system has similar functionality to one of the case studies in [Wul97].[3]

As usual in this sort of analysis, it suffices to look at only two security levels of user, on the grounds that more complex security policies can be partitioned into multiple binary analyses: see [RWW96, Ros95,

[3] [Wul97] is concerned with abstractions and using them to reason about information flow and computer security; it does not deal with the Parameterised Verification Problem.

Wul97, Ros98] for details. The system thus works with two user identities: a high security user `Hugh` and a low security user `Lois`.

Without going into details, the system consists of a manager process `DBM` and a disk process `Disk`. It has two variable types, `RECORDS` and `DATA`, and it uses an array with indices from `RECORDS` and values from `DATA` to model the contents of the disk. It also uses a 2-valued predicate `rlevel :: RECORDS -> {0, 1}` which provides the security level associated with each record, and it has a constant `inval :: DATA` for the initial value of all the records.

That the system is secure is understood to mean that there can be no information flow from `Hugh` to `Lois` across it. As we have remarked in Section 3, this can be verified by verifying that a suitable abstraction of the system, namely what it looks like to `Lois` when `Hugh`'s actions are turned into internal nondeterminism, is deterministic [RWW96, Ros95, Wul97, Ros98].

Let us call this abstraction `P`. It turns out that `P` satisfies all the assumptions of Theorem 1 with `X`, `Y`, `r` and `c` now being called `RECORDS`, `DATA`, `rlevel` and `inval` respectively, and that

$$B = 2 \times (\texttt{limit(Hugh)} + \texttt{limit(Lois)}) + 1$$

where `limit(Hugh)` and `limit(Lois)` are limits of how many records the users can have open or locked at any one time. Thus Theorem 1 has successfully reduced the problem of verifying that the system is secure for each instantiation of `RECORDS`, `DATA`, `rlevel` and `inval` to verifying that it is secure for a single finite instantiation.

For a few small values of `limit(Hugh)` and `limit(Lois)`, we have verified on the model checker FDR that the reduced instantiation of `P` is indeed deterministic. It remains an open problem, outside the scope of this paper, to automatically establish the determinism for any `limit(Hugh)` and `limit(Lois)` and when one or both of those limits are removed.

A complete CSP_M script with the case study can be found at `http://www.comlab.ox.ac.uk/oucl/publications/books/concurrency/examples`.

6 Further Work

We should stress that the techniques we have developed are not restricted to verifying determinism. Indeed, we hope to obtain similar theorems for verifying arbitrary conditions. In CSP, that means for verifying that a concurrent system which uses unbounded arrays refines [Hoa85, Ros98] a given specification.

Acknowledgements

We are very grateful to Stephen Brookes, Michael Goldsmith, Ramin Hojati, Dominic Hughes, Martin Hyland, Norris Ip, Gavin Lowe, Oege de Moor, Kedar Namjoshi, Peter O'Hearn, Luke Ong, Irfan Zakiuddin and others for helpful discussions, and to Lars Wulf for basing the functionality of our case study on a case study in his thesis.

References

[AK86] Apt, K.R. and D.C. Kozen, *Limits for Automatic Verification of Finite State Concurrent Systems*, 307–309, Information Processing Letters 22 (6), 1986.

[CE+96] Clarke, E.M., R. Enders, T. Filkorn and S. Jha, *Exploiting Symmetry in Temporal Logic Model Checking*, 77–104, in [Eme96].

[CGJ95] Clarke, E.M., O. Grumberg and S. Jha, *Verifying Parameterized Networks Using Abstraction and Regular Languages*, 395–407, Proc. of the 6^{th} CONCUR, Springer LNCS 962, 1995.

[CZ+97] Corella, F., Z. Zhou, X. Song, M. Langevin and E. Cerny, *Multiway Decision Graphs for Automated Hardware Verification*, 7–46, Formal Methods in System Design 10 (1), Kluwer, 1997.

[Eme96] Emerson, E.A. (ed.), Formal Methods in System Design 9 (1–2) (Special Issue on Symmetry in Automatic Verification), Kluwer, 1996.

[EN96] Emerson, E.A. and K.S. Namjoshi, *Automatic Verification of Parameterized Synchronous Systems*, 87–99, Proc. of the 8^{th} CAV, Springer LNCS 1102, 1996.

[ES96] Emerson, E.A. and A.P. Sistla, *Symmetry and Model Checking*, 105–131, in [Eme96].

[FS97] Formal Systems (Europe) Ltd, *Failures-Divergence Refinement: FDR2 Manual*, 1997.

[Gra94] Graf, S., *Verification of a Distributed Cache Memory by Using Abstractions*, 207–219, Proc. of the 6$^{\text{th}}$ CAV, Springer LNCS 818, 1994.

[GS92] German, S.M. and A.P. Sistla, *Reasoning About Systems with Many Processes*, 675–735, Journal of the ACM 39 (3), 1992.

[HB95] Hojati, R. and R.K. Brayton, *Automatic Datapath Abstraction in Hardware Systems*, 98–113, Proc. of the 7$^{\text{th}}$ CAV, Springer LNCS 939, 1995.

[HDB97] Hojati, R., D.L. Dill and R.K. Brayton, *Verifying Linear Temporal Properties of Data Insensitive Controllers Using Finite Instantiations*, Proc. of the 13$^{\text{th}}$ CHDL, 1997.

[HGD95] Hungar, H., O. Grumberg and W. Damm, *What if Model Checking Must be Truly Symbolic?*, 230–244, Proc. of the 1$^{\text{st}}$ TACAS, BRICS Notes Series NS-95-2, Department of Computer Science, University of Aarhus, 1995. (Also in Springer LNCS 1019.)

[HIB97] Hojati, R., A.J. Isles and R.K. Brayton, *Automatic State Reduction Technique for Hardware Systems Modelled Using Uninterpreted Functions and Infinite Memory*, Proc. of the 2$^{\text{nd}}$ IEEE HLDVT, 1997.

[HL95] Hennessy, M. and H. Lin, *Symbolic Bisimulations*, 353–389, Theoretical Computer Science 138 (2), 1995.

[HM+95] Hojati, R., R. Mueller-Thuns, P. Lowenstein and R.K. Brayton, *Automatic Verification of Memory Systems which Execute Their Instructions Out of Order*, Proc. of CHDL, 1995.

[Hoa85] Hoare, C.A.R., *Communicating Sequential Processes*, Prentice Hall, 1985.

[ID96a] Ip, C.N. and D.L. Dill, *Better Verification Through Symmetry*, 41–75, in [Eme96].

[ID96b] Ip, C.N. and D.L. Dill, *Verifying Systems with Replicated Components in Murφ*, 147–158, Proc. of the 8$^{\text{th}}$ CAV, Springer LNCS 1102, 1996.

[Jen96] Jensen, K., *Condensed State Spaces for Symmetrical Coloured Petri Nets*, 7–40, in [Eme96].

[JP93] Jonsson, B. and J. Parrow, *Deciding Bisimulation Equivalences for a Class of Non-finite-state Programs*, 272–302, Information and Computation 107 (2), 1993.

[KM+97] Kesten, Y., O. Maler, M. Marcus, A. Pnueli and E. Shahar, *Symbolic Model Checking with Rich Assertional Languages*, 424–435, Proc. of the 9$^{\text{th}}$ CAV, Springer LNCS 1254, 1997.

[Laz98] Lazić, R.S., D.Phil. Thesis, Oxford University Computing Laboratory, to appear in 1998.

[LR96] Lazić, R.S. and A.W. Roscoe, *Using Logical Relations for Automated Verification of Data Independent CSP*, Proc. of the Workshop on Automated Formal Methods (Oxford, U.K., June 1996), to appear in Electronic Notes in Theoretical Computer Science.

[LR98a] Lazić, R.S. and A.W. Roscoe, *Verifying Determinism of Concurrent Systems Which Use Unbounded Arrays*, Technical Report, Oxford University Computing Laboratory, 1998.

[LR98b] Lazić, R.S. and A.W. Roscoe, *A Semantic Study of Data Independence with Applications to Model Checking*, to be submitted to Theoretical Computer Science in 1998.

[Ros95] Roscoe, A.W., *CSP and Determinism in Security Modelling*, Proc. of the IEEE Symposium on Security and Privacy, 1995.

[Ros98] Roscoe, A.W., *The Theory and Practice of Concurrency*, Prentice Hall, 1998.

[Rus97] Rushby, J.M., *Are Formal Methods Best Used for Refutation or for Verification?*, A talk given at the 4$^{\text{th}}$ NASA Langley Formal Methods Workshop (LFM), 1997. (Not in the published proceedings.)

[RWW96] Roscoe, A.W., J.C.P. Woodcock and L. Wulf, *Non-interference Through Determinism*, 27–54, Journal of Computer Security 4 (1), 1996. (Revised from Proc. of ESORICS '94, Springer LNCS 875.)

[Sca98] Scattergood, J.B., *Tools for CSP and Timed CSP*, D.Phil. Thesis, Oxford University Computing Laboratory, to appear in 1998.

[VB+97] Velev, M., R.E. Bryant and A. Jain, *Efficient Modelling of Memory Arrays in Symbolic Simulation*, 388–399, Proc. of the 9$^{\text{th}}$ CAV, Springer LNCS 1254, 1997.

[Wol86] Wolper, P., *Expressing Interesting Properties of Programs in Propositional Temporal Logic*, 184–193, Proc. of the 13$^{\text{th}}$ ACM POPL, 1986.

[Wul97] Wulf, L., *Interaction and Security in Distributed Computing*, D.Phil. Thesis, Oxford University Computing Laboratory, 1997.

[YJL96] Yen, H.-C., S.-T. Jian and T.-P. Lao, *Deciding Bisimulation and Trace Equivalences for Systems With Many Identical Processes*, 445–464, Theoretical Computer Science 170 (1–2), 1996.