*Electronic Notes in Theoretical Computer Science 50 No. 4 (2001) { Proc. VEPAS 2001 URL:* [*http://www.elsevier.nl/locate/entcs/volume50.html*](http://www.elsevier.nl/locate/entcs/volume50.html) *15 pages*

*Networks of Processes with Parameterized State Space*

*K. Baukus K. Stahl*

*Institute of Computer Science and Applied Mathematics CAU Kiel, Preusserstr. 1{9, D-24105 Kiel, Germany.* [*kba@informatik.uni-kiel.de*](mailto:kba@informatik.uni-kiel.de)

[*kst@informatik.uni-kiel.de*](mailto:kst@informatik.uni-kiel.de)

*S. Bensalem Y. Lakhnech Verimag, Centre Equation, 2 Av. de Vignate,*

*38610 Gi eres, France.* [*bensalem@imag.fr*](mailto:bensalem@imag.fr)[*lakhnech@imag.fr*](mailto:lakhnech@imag.fr)

*Abstract*

*In general, the veri cation of parameterized networks is undecidable. In recent years there has been a lot of research to identify subclasses of parameterized systems for which certain properties are decidable. Some of the results are based on nite ab- stractions of the parameterized system in order to use model-checking techniques to establish those properties. In a previous paper we presented a method which allows to compute abstractions of a parameterized system modeled in the decidable logic WS1S. These WS1S systems provide an intuitive way to describe parameterized systems of nite state processes. In practice however, the processes in the network themselves are in nite because of unbounded data structures. One source of un- boundedness can be the usage of a parameterized data structure. Another typical source may be the presence of structures ranging over subsets of participating pro- cesses. E.g., this is the case for group membership or distributed shared memory consistency protocols. In this paper we use deductive methods to deal with such networks where the data structure is parameterized by the number of processes and an extra parameter. We show how to derive an abstract WS1S system which can be subject to algorithmic veri cation. For illustration of the method we verify the correctness of a distributed shared memory consistency protocol using PVS for the deductive veri cation part and the tools pax and SMV for the algorithmic part.*

*c 2001 Published by Elsevier Science B. V. Open access under C*[*C BY-NC-ND license*](http://creativecommons.org/licenses/by-nc-nd/3.0/)*.*

# *1 Introduction*

*Networks with an a priori unknown number of nodes occur everywhere in prac-* tice, e.g., processes running on a computer, machines in a token ring, a LAN or even the world wide web itself. Therefore, a lot of algorithms like mutual exclusion, leader election, group membership, or distributed memory algo- rithms are developed for such parameterized networks and they are expected to work properly for every concrete number of participants. Hence, although the problem is known to be undecidable (Apt and Kozen, [2]), there has been much interest in the veri cation of such protocols. Also, automated and semi- automated methods for the veri cation of restricted classes of parameterized networks have been developed.

*Deductive methods presented in [18,25,5,23,14,19] are based on induction* on the number of processes. These methods require nding a network invariant that abstracts any arbitrary number of processes with respect to a pre-order that preserves the property to be veri ed.

*Algorithmic methods presented in [13,10,11] show that for classes of ring* networks of arbitrary size and client-server systems, there exists k such that the veri cation of the parameterized network can be reduced to the veri cation of networks of size up to k.

*A semi-automated method presented in [17] uses the idea of representing* sets of states of parameterized networks by regular languages, where addi- tionally nite-state transducers are used to compute predecessors. In [1,16] acceleration techniques are applied to consider the e ect of taking in nitely often a transition.

*In [3] we showed how to represent parameterized systems in the decidable* logic WS1S, i.e., the current state of the system is modeled as a xed number of

*nite subsets of the natural numbers and the transitions of the processes in the* network are described in WS1S. Given a boolean abstraction relation in WS1S this allows to construct the abstract system automatically. The method was used to verify several parameterized protocols. With an additional marking algorithm and the lifting of fairness conditions ([4]) we were able to establish liveness properties for these protocols as well.

*The method is implemented in a tool called pax 1 , that uses the decision* procedures of Mona [15] to check the satis ability of WS1S formulae.

*In this paper we show how to combine this method with deductive veri -* cation in order to verify protocols which do not ful ll the above requirements, i.e., each process in the network has an unbounded state space. Here, we ad- dress di erent reasons for the unboundedness. First, we deal with the problem of having two parameters; one gives us the number of processes, the other pa- rameterizes each process itself. Second, it is often the case that the processes render with data structures to keep information about the other processes,

*1* [*http://www.informatik.uni-kiel.de/~kba/pax*](http://www.informatik.uni-kiel.de/~kba/pax)

*i.e., they have data structures ranging over subsets of the involved processes* and are therefore unbounded.

*We illustrate our method using a typical example: a distributed shared* memory protocol. The protocol has two parameters, the number of processes and the number of memory pages. In order to guarantee exclusive write and multiple read the processes have to know the actual access privileges of the other processes. We give a generic abstraction for this type of protocols in the framework of PVS and show how to analyze the abstract WS1S system using pax and SMV to prove safety and liveness properties of the protocol.

# *2 Protocol Description*

*To illustrate our method we verify a distributed shared memory consistency* protocol by Li and Hudak presented in [20]. The idea of distributed shared memory (DSM) is to allow processors in a distributed environment to utilize each other's local memories. DSM systems provide a virtual address space for a network of processors. They replicate or migrate data across the network to handle requests of threads running on the processors.

*Li and Hudak's protocol is a multiple-reader, single-writer protocol; several* processes are allowed to have read access to the same data while write access is exclusive. The data which is subject of the read/write requests is organized into pages. A page table on each processor maintains the current access status of the processor for each page. The status may be read or write and keeps the information whether the processor owns the page. The owner is the processor last having write privileges to a page. Moreover, the status may be nil if the processor has no local copy of the page or if the page has been modi ed by some other processor.

*When a processor wants to upgrade its privileges (from nil to read or from* read to write) it sends a corresponding request via broadcast. The owner handles the request; for a read request it adds the requesting processor to the copy-set, the list of processors with read access. Processors in the copy- set have to be informed when a processor wants write access. Each of these processors has to acknowledge that it changed its page table for that page to nil. Then, the ownership changes to the requesting processor. The pseudo- code in Figure 1 describes the various actions [12].

*As communication mechanism we assume to have one central request queue* where all the requests are sent to. Acknowledgments are sent directly to the requesting processors. The description of the protocol immediately gives us four veri cation goals:

*Exclusive Ownership: For each page p there is always at most one owner.*

*Exclusive Write: Whenever there is a processor having write access for p* then there is no processor with read access.

|  |  |
| --- | --- |
| *req rd(p)*  *PTable[p].lock:=true; broadcast read request for p;* | *send rd ack(p,i) PTable[p].lock:=true;*  *IF PTable[p].owner THEN PTable[p].copyset:=PTable[p].copyset [ fig; PTable[p].access:=read;*  *send ack and p to i; FI*  *PTable[p].lock:=false;* |
| *get rd ack(p) PTable[p].access:=read; PTable[p].lock:=false;* |
| *req wr(p)*  *PTable[p].lock:=true; broadcast write request for p;* | *send wr ack(p,i) PTable[p].lock:=true;*  *IF PTable[p].owner THEN send p and copyset to i; PTable[p].access:=nil; PTable[p].owner:=false; FI*  *PTable[p].lock:=false;* |
| *get wr ack(p,copy-set) PTable[p].access:=inv;*  *req invalidate(p,copyset);* |
| *req invalidate(p, copyset)*  *FOR k in copyset DO*  *send inv request for p to k;* | *send inv ack(p)*  *IF PTable[p].access=read THEN PTable[p].access:=nil;*  *send acknowledgment; FI* |
| *read nil ack(i) PTable[p].copyset:=*  *PTable[p].copyset n fig;* |
| *get owner*  *IF PTable[p].copyset=; AND PTable[p].access=inv THEN*  *PTable[p].access:=write; PTable[p].copyset:=;; PTable[p].owner:=true; PTable[p].lock:=false; FI* | |

*Fig. 1. Pseudo-code of one processor according to the DSM protocol*

*Copy-Set Adequacy: Processors having read access to page p are always in* the copy-set of the owner of p except for the owner itself.

*Liveness: Whenever a processor requests access privileges it eventually ob-* tains them.

# *3 Veri cation by Abstraction*

*First, let us recall some de nitions and the idea of proving properties of sys-* tems by abstraction. Given a deadlock-free 2 transition system S = (V; ; T ) and a total abstraction relation A, we say that SA = (VA; A; TA) is an abstraction of S w.r.t. , denoted by S v SA, if the following conditions are satis ed: (1)1( ) and (2) Æ11 Æ for corresponding

*A A*

*2 T , A 2 TA.*

*In case A is nite, we call nite abstraction relation. Let '; 'A be* LTL formulae and let [[']] (resp. [['A]]) denote the set of models of ' (resp. ' ). Then, from S v S ,1([[' ]]) [[']], and S j= ' we can conclude S j= '. This statement, which is called preservation result, shows the interest of veri cation by abstraction: since if SA is nite, it can automatically be checked whether SA j= 'A. In fact, a similar preservation result holds for any temporal logic without existential quanti cation over paths, e.g., 8CTL?, LTL, or 2 [7,8,21].

*A A A A A*

*If we have already proven some state property ' to be invariant in S, i.e.,* S j= 2', we can strengthen condition (2) to (20)

*( \ f(s ;s ) j s j= '; s j= 'g) Æ11 Æ :*

*0 1 0 1 A*

*This allows to search for some smaller abstract system in order to establish* properties for the concrete system. We denote this type of abstraction by S v' SA.

*In case S is a fair transition system with F as fairness formula and if FA* is the fairness formula of S , then by requiring1([[:F ]]) [[:F]], we have the same preservation result as above. We indicate this type of abstraction by S vF SA.

*A A*

*3.1 System Reduction using PVS*

*Our aim is to verify the DSM protocol by Li and Hudak using abstraction* techniques. In [3] we presented how to compute abstractions automatically for networks with nite processes modeled in the decidable logic WS1S. Unfor- tunately, Li and Hudak's protocol handles a parameterized number of pages (beside the parameterized number of processors) and each page table entry contains a set (copy-set) of processor indices.

*To reduce the state space of the processors we introduce a global-copy (see* Figure 2 for the PVS declaration). This variable is global to all processors. We added assignments to it whenever the processor having the ownership for one page updates its local copy-set. If we can prove that the local copy-set of some processor, whenever needed, coincides with the global-copy we can get rid of all the local copy-sets.

*2 Throughout this paper we only consider deadlock free transition systems which can be achieved by adding an idle transition.*

*Access: type = fwrite, nil, read, invalidateg Requests: type = fread req, write reqg*

*Direct: type = fwrite ack, read ack, nil reqg N , P : posnat*

*Processor: type = below(N ) Page: type = below(P )*

*Copy set type: type = setof[Processor]*

*Req channel: type = list[[Processor, Requests, Page]] Direct channels: type = [Processor ! list[[Direct, Page]]] Nil Acks: type = [Page ! list[Processor]]*

*PageEntry: type =*

*[# access: [Processor ! Access],*

*owner?, locked?: [Processor ! bool], copy set: [Processor ! Copy set type], send copy, global copy: Copy set type #]*

*State: type =*

*[# PageTable: [Page ! PageEntry], Reqs: Req channel,*

*DirectCom: Direct channels, Nils: Nil Acks #]*

*Fig. 2. Parts of PVS theory*

*To deal with the second parameter (the page parameter) we identify a* class of parameterized networks for which we can verify certain properties by instantiating the second parameter with 1.

*These intermediate steps give us a reduced system presentable in WS1S* but still allows to prove properties for the original protocol. We use PVS [22] to establish the reduction. Since PVS allows to use higher order logic for speci cations it is straightforward to give a PVS model for the pseudo-code in Figure 1. Figure 3 shows an example transition expressed as a relation between pre- and post-state.

*Using PVS it was straightforward to prove that indeed the global-copy* matches with the local copy-set whenever needed. The property (see Figure 4) is inductive over all transitions and initially ful lled and therefore invariant.

*Generality of this step*

*As already mentioned network protocols often handle data structures rang-* ing over sets of processors in the network. As examples we listed shared mem- ory protocols such as the one described in Section 2 or group membership protocols. To illustrate how the idea to reduce the state space of each pro- cessor by introducing a global structure used by all processors can be applied to other protocols, let us consider a group membership protocol. Brie y, each

*s, s1: var State*

*i: var Processor p: var Page*

*req rd(s, s1, i, p): bool =*

*access(PageTable(s)(p))(i) = nil ^*

*: locked?(PageTable(s)(p))(i) ^*

*s1 = s with [(PageTable)(p) := PageTable(s)(p)*

*with [(locked?)(i) := true],*

*Reqs := cons((i, read req, p), Reqs(s))] Fig. 3. Requesting read access*

*Global Local(s, i, p): bool =*

*(owner?(PageTable(s)(p))(i)*

*copy set(PageTable(s)(p))(i) = global copy(PageTable(s)(p)))*

*Fig. 4. Equality of global-copy and copy-set for owner*

*processors keeps track which processors are still alive and vital part of the net-* work. Due to continuous communication errors are detected and processors are removed from the membership-list of the well functioning processors. Two properties are of interest for those protocols; agreement, meaning that the well functioning processors agree on their membership-lists, and validity, meaning that an error will be detected eventually and then the membership-lists corre- sponds to the set of well functioning processors. We analyzed a synchronous group membership protocol by proving rst agreement deductively. Then, we could reduce the system by using a global membership-list maintained by the processors working properly.

*Now, back to our DSM protocol. Even if we remove all local copy-sets we* have a system of processors where each of them maintains a page table for a parameterized number of pages. But, we observe (and can prove with PVS) that all transitions alter only the page entry for exactly one page and consume or produce only messages concerning this page. We x these assumptions in the next de nition.

*De nition 3.1 Let S(N; P ) be a parameterized system with N processors* and a second parameter P . Let the processors communicate over some message queues q1 ;::: ; qk where each message is of the form (i; msg; p) with i < N, p < P , and msg of some nite type M. The state space of each processor j is an array a[j][0::P 1] of size P and of nite type T .

*We call S(N; P ) strictly parameterized in P if each processor has initially* the same con guration for each page:

*8i < N : 8p1 ; p2 < P : a[i][p1 ] = a[i][p2 ] ^ ^ql = hi ;*

*k*

*l=1*

*and each transition in S(N; P ) is either*

*an internal step a[i][p] = t ^ a0 = a([i][p] 7! t0 ) ^ Vk*

*q0 = q*

*1*

*or a communication step*

*1 l=1 l l*

*a[i][p] = t1 ^ a[j][p] = t2 ^ a0 = a([i][p] 7! t0 ; [j][p] 7! t0 ) ^*

*1 2*

*[ql = hj; msg ; pi q0 ^ msg = m ^ ][q0 = qo hj; msg ; pi^ ]*

*l*

*o*

*1*

*1*

*2*

*^*

*q*

*r*

*r6=l;o*

*0 = qr ;*

*where t1 ; t2; t0 ; t0*

*are constants in T and m; msg1; msg2 are constants in M.*

*1 2*

*Concerning the communication step either the message consuming part ql =*

*hj; msg1; pi q0 or the message producing part q0 = qo hj; msg2; pi may be empty.*

*l o*

*We call such transitions changing only array entries for p and handling only*

*p-messages, p-transitions.* *2*

*As usual we denote with [[S(N; P )]] the set of computations = s0; s1;:::*

*of S(N; P ). Now, let [[S(N; P )]]*

*denote those computations p = sp; sp;:::*

*p 0* *1*

*derived from the original ones by projecting the array a[0::N 1][0::P 1] to*

*the array entry a[0::N 1][p] and projecting the contents of the communication* channels to messages with tag p. Then, we can show:

*Lemma 3.2 For a system S(N; P ) strictly parameterized in P we have for* all P and p < P

*[[S(N; P )]] p = [[S(N; P )]] 1 = [[S(N; 1)]] ;*

*where [[ ]] denotes the set of computations with arbitrary interleaved idle tran-* sitions.

*Proof. ' rst equality': Concerning one processor i each computation starts* with a[i][p] = a[i][m] = t for t 2 T (N) and p; m < P . The message queues are empty. Hence, the same transitions for p and m are enabled which produce the same messages (msg; p) resp. (msg; m) and yield the same array entry t0. Hence, in each computation we can exchange p- and m-transitions. This gives us the rst equality.

*'second equality': If we consider in S(N; P ) only 1-transitions to happen,* trivially we have [[S(N; 1)]] [[S(N; P )]]. Since all p-transitions with p 6= 1 do not alter a[i][1] for any processor i and do not consume or produce 1-messages they are idle transitions in [[ ]] 1. Hence, we have [[S(N; P )]] 1 [[S(N; 1)]] .2

*The introduction of idling transitions is needed because we have made no* assumptions about fairness. Hence, it is not guaranteed that some p tran- sitions occur or when they occur as long as other transitions are enabled. Nevertheless concerning invariance properties we get immediately:

*Corollary 3.3 Let S(N; P ) a system strictly parameterized in P . Let '(p)* be a state formula. Then, we have:

*S(N; 1) j= 2'(1) , S(N; P ) j= 8p < P : 2'(p) :*

*2*

*To deal with liveness properties we need some assumptions about fairness.* We call S(N; P ) weak fair if all transitions continuously enabled from a certain point in a computation are eventually taken. If moreover S(N; 1) never blocks a queue, i.e., messages in the queues are eventually consumed, we can deduce from Lemma 3.2:

*Corollary 3.4 Let S(N; P ) be a system strictly parameterized in P and as-* sume S(N; P ) is weak fair. Let '(p) be a property expressed in LTL/X. Then, if S(N; 1) never blocks a queue we have:

*S(N; 1) j= '(1) , S(N; P ) j= 8p < P : '(p) :*

*2*

*Now, we observe that our reduced system is indeed strictly parameterized* in P . Hence, we can get rid of the second parameter and are prepared to represent the resulting network in the framework of WS1S. Then, we use abstraction techniques explained in the next section to analyze it.

*3.2 Veri cation using pax*

*First we brie y recall the de nition of weak second order theory of one suc-* cessor (WS1S for short) [6,24].

*Terms of WS1S are built up from the constant 0 and 1st-order variables by* applying the successor function suc(t) (\t + 1"). Atomic formulae are of the form b, t = t0, t < t0, t 2 X, where b is a boolean variable, t and t0 are terms, and X is a set variable (2nd-order variable). WS1S formulae are built up from atomic formulae by applying the boolean connectives as well as quanti cation over both 1st-order and 2nd-order variables.

*WS1S formulae are interpreted in models that assign nite sub-sets of !* to 2nd-order variables and elements of ! to 1st-order variables. The interpre- tation is de ned in the usual way.

*Given a WS1S formula f , we denote by [[f ]] the set of models of f . The* set of free variables in f is denoted by free(f ).

*Finally, we recall that by Buchi [6] and Elgot [9] the satis ability problem* for WS1S is decidable. Indeed, the set of all models of a WS1S formula is representable by a nite automaton (see, e.g., [24]).

*Now, we introduce WS1S transition systems which are transition systems* with variables ranging over nite sub-sets of ! and show how they can be used to represent the parameterized system from the previous section.

*De nition 3.5 [WS1S Transition Systems] A WS1S transition system S =* (V; ; T ) is given by the following components:

*V = fX1;::: ; Xkg: A nite set of second order variables where each variable* is interpreted as a nite set of natural numbers.

*: A WS1S formula with free( ) V describing the initial condition of* the system.

*T : A nite set of transitions where each 2 T is represented as a WS1S* formula (V; V0), i.e., free( ) V [ V0. We use the primed version of the variables to denote the post-state. *2*

*The computations of S are de ned as usual. Moreover, let [[S]] denote the set* of computations of S.

*In the previous section we have proven that we can replace the local vari-* ables copy-set of each processor by one global variable global-copy. This and the argument that allows us to verify the system instantiated with only one page concerning the page parameter gives us the possibility to model the re- duced system as a WS1S system.

*As set of second order variables we choose*

*V = f Procs; Read ; Write; Invalidate; Owner ; Locked ; Global Copy;*

*Read Req ; Write Req; Nil Req; Read Ack ; Write Ack ; Nil Ack g :*

*We have no set to represent the processors being in state nil. We assume those* which are not in read, write, and invalidate to have nil access to the page. In WS1S it is not possible to represent queues, hence, we model them as sets. This, of course, gives us an abstraction. On the other hand we lose some fairness, e.g., when a request of processor i is in the queue, then the request is eventually granted when read by the owner, or the owner eventually stops to handle any request:

*2(i 2 Read Req ) 3(i 2 Read Ack ) \_ 32:send rd ack)*

*Hence, we add those fairness conditions F to the WS1S system. The initial* condition of our WS1S system reads as follows:

*(9n : 8i : i < n ) i 2 Procs)*

*^ (9j : j 2 Procs ^ Read = fjg^ Owner = fjg)*

*^ Locked = ;^ Write [ Invalidate = ;*

*^ Read Req [ Write Req [ Nil Req = ;*

*^ Read Ack [ Write Ack [ Nil Ack = ; :*

*To illustrate how the transitions of our PVS speci cation (simpli ed by* reduction to one page) can be expressed in WS1S we give here req rd:

*9i : i 2= Read [ Write [ Invalidate [ Locked*

*^ Locked 0 = Locked [ fig^ Read Req0 = Read Req [ fig*

*^ ^*

*X2VnfRead Req ;Locked g*

*X0 = X :*

*Next, we want to analyze the WS1S system de ned above by abstraction.* Therefore, we apply the methods presented in [3,4]. Let S = (V; ; T ) be a given WS1S system and let be a boolean abstraction function expressed

*as a WS1S formula b(V; VA). Since the abstract variables are booleans, the*

*abstract system we construct is nite, and hence, can be subject to model-* checking techniques. Moreover, we make use of the fact that both (V; VA) and the transitions in T are expressed in WS1S to give an e ective construction of the abstract system.

*b*

*Hence, the initial states of the abstract system we construct can be de-* scribed by the formula

*9V : (V; VA) :*

*b*

*As transitions the abstract system contains for each concrete transition an* abstract transition A, which is characterized by the formula

*9V; V0 : (V; VA) ^ (V; V0) ^ (V0; V0 )*

*A*

*b b*

*with free variables VA and V0 . It is obvious that those de nitions satisfy con-* ditions (1) and (2) from Section 3. If we have already proven some invariance property to hold for S, we can choose

*A*

*9V; V0 : (V) ^ b(V; VA) ^ (V; V0) ^ b(V0; V0 ) ^ (V0)*

*A*

*as abstract transitions which satisfy condition (20).*

*To compute the abstract system, one has to nd all states ful lling these* formulae, which is possible since they are WS1S formulae. This means, choos- ing properties 'i(V) of the concrete system as abstract variables ai , 'i(V) al-

*lows us to compute automatically the abstract system according to the boolean*

*abstraction function Vn a , ' (V).*

*i=1*

*i* *i*

*Sometimes, we are interested in so-called universal progress or response* properties, that is, properties that guarantee each single process i eventually makes some progress, or each request by i to j eventually gets a response by j. To prove those properties by abstraction the abstraction function has to focus on processors, i.e., the abstraction function contains i or i; j as free variables ( (V; VA; i) or (V; VA; i; j)).

*b b*

*Then, the abstract system contains as abstract transitions*

*9V; V0 : 9i; j : b(V; VA; i; j) ^ (V; V0) ^ b(V0; V0 ; i; j)*

*A*

*(or those with invariance constraints) and starts in initial state:*

*9V : 9i; j : (V; VA; i; j) :*

*b*

*Both methods are implemented in pax and we use them to analyze our system.*

# *4 Veri cation Results*

*Using PVS it was easy to prove some intuitive properties to be inductive* over all transitions and therefore being invariant. These properties state that the processor having write privileges for some page is also the owner, that ownership is exclusive, and that the owner knows the processors with read access (see Figure 5). Hence, we have already two of our four properties, namely exclusive ownership and copy-set adequacy. As stated in Section 3 we can use these invariants to strengthen our WS1S system.

*Invariant1(s, i, p): bool =*

*access(PageTable(s)(p))(i) = write*

*: locked?(PageTable(s)(p))(i) ^ owner?(PageTable(s)(p))(i)*

*Invariant2(s, p): bool = (9 (i: Processor):*

*owner?(PageTable(s)(p))(i)*

*(8 (j: Processor): : (j = i) ^ : owner?(PageTable(s)(p))(j)))*

*Invariant3(s, p): bool =*

*8 (j: Processor): 9 (i: Processor):*

*(access(PageTable(s)(p))(j) = read ^*

*: owner?(PageTable(s)(p))(j) ^ owner?(PageTable(s)(p))(i))*

*(j 2 copy set(PageTable(s)(p))(i))*

*Fig. 5. Invariants proven with PVS*

*Using pax we can successively check for more invariants. It is possible to* strengthen the exclusive ownership property to

*Owner \ Write Ack = ;*

*^ Owner \ Invalidate = ;*

*^ Write Ack \ Invalidate = ;*

*^ 9j : fjg = Owner [ Write Ack [ Invalidate*

*which states that to grant a write request the owner gives away the ownership* of the page to acknowledge the request. The acknowledgment causes the requesting processor to go in the invalidate state to inform all processors having read privileges. We take the property given above as the de nition for one abstract variable (or we can take one variable for each conjunct). The resulting state space of the abstract system has exactly one state, namely the state where the variable is (or all variables are) true.

*The rst of the remaining two properties, exclusive write access, can be* expressed as:

*(1) 8i; j : 2(i 2 Write ) j 2= Read )*

*This is because we have already proven exclusive ownership and that the* processor with write privileges is the owner. The second one, that a request will be eventually granted, is described by the LTL formula:

*(2) 8i : 2(i 2 Write Req ) 3(i 2 Write ^ i 2 Owner ))*

*Both properties are universal as de ned in the previous section, hence, we* de ne an abstraction function concentrating on two processors i; j:

*(V; VA; i; j) i 6= j ^*

*^ aX , i 2 X ^*

*i*

*X2V*

*^ a , j 2 X*

*X*

*j*

*X2V*

*The pax tool generates a nite abstract system and translates it to the SMV* input language. Moreover, pax automatically adds new boolean variables to the SMV speci cation for each transition to monitor which transition was taken in the last step.

*SMV veri es property 1 in about half a second. But SMV fails to prove* property 2. It generates a counter example which loops with the transitions requesting for read or write access. At the concrete level this is not possible, in fact, each processor locks the page when performing a request. The next request by that processor can only be done after unlocking the page which cor- responds to receiving the requested privileges. Hence, in the concrete system we can have only as much requests as processors. In [4] we presented a mark- ing algorithm to transform such dependencies into liveness conditions stating that a transition decreasing some set can only be taken in nitely often when another one increasing the same set is also taken in nitely often. Applied to the set Procs n Locked that gives us the fairness condition:

*23(req rd \_ req wr) ) 23(get rd ack \_ get wr ack)*

*Note, the generated fairness conditions are guaranteed to hold in the concrete* system and are therefore not part of its fairness requirements F. In contrast, we also need some fairness condition FA at the abstract level corresponding to the weak fairness assumptions F in the concrete system as de ned in Sec- tion 3. Taking these fairness conditions together with the generated ones as assumptions to constrain our abstract system, SMV needs 15 seconds to es- tablish property 2. By Corollary 3.4 we have established the correctness of the original distributed shared memory protocol.

# *5 Conclusions*

*We have shown how to represent parameterized networks where each proces-* sor has an unbounded state space in the framework of PVS. The use of higher order logic allows an intuitive modeling. For illustration of our method we have chosen a distributed shared memory protocol which is representative for a class of network protocols which maintain lists of other processors in the network. This makes the local state space of the processors unbounded. We used theorem proving to establish trace equality with another system main- taining only one global unbounded data structure. Besides the parameter specifying the number of participating processors another parameter speci es the data structures handled by the processors. We identi ed a subclass of doubly-parameterized systems for which we can reduce the veri cation task by analyzing the system instantiated with 1 as the data structure parameter. Hence, we were able to describe the reduced system as a WS1S system and apply the pax tool to verify the systems properties by abstraction. Given the abstraction function { which is straight forward { the computation of the ab- stract system is fully automatically. The abstract system is nite state and can be subject to model-checking techniques. The properties proven also include

*liveness properties which needed some extra fairness conditions generated by* a marking algorithm and are guaranteed to hold in the concrete system.

*While our method is not fully automatic and needs user interaction, it is far* from pure theorem proving. Once we have come up with the WS1S system, generating abstractions and checking properties for the abstract system is an algorithmic task. Also, the veri cation of liveness properties by theorem proving would require a lot more user interaction than identifying fairness conditions which can be generated by the marking algorithm to exclude the counterexamples produced by the model-checker.

# *References*

*[1] P.A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling Global Conditions in Parameterized System Veri cation. In N. Halbwachs and*

*D. Peled, editors, CAV '99, volume 1633 of LNCS, pages 134{145. Springer, 1999.*

*[2] K. Apt and D. Kozen. Limits for Automatic Veri cation of Finit-State Concurrent Systems. Information Processing Letters, 22(6):307{309,* *1986.*

*[3] K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl. Abstracting WS1S Systems to Verify Parameterized Networks. In S. Graf and M. Schwartzbach, editors, TACAS'00, volume 1785, pages 188 { 203. Springer, 2000.*

*[4] K. Baukus, Y. Lakhnech, and K. Stahl. Verifying Universal Properties of Parameterized Networks. In M. Joseph, editor, FTRTFT'00, volume 1926, pages 291 { 304. Springer, 2000.*

*[5] M.C. Browne, E.M. Clarke, and O. Grumberg. Reasoning about networks with many identical nite state processes. Information and Computation, 1989.*

*[6] J.R. Buchi. Weak Second-Order Arithmetic and Finite Automata. Z. Math. Logik Grundl. Math., 6:66{92, 1960.*

*[7] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. ACM Transactions on Programming Languages and Systems, 16(5), 1994.*

*[8] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems: Abstractions preserving ACTL, ECTL and CTL* *In E.-R. Olderog,*

*editor, Proceedings of PROCOMET '94. North-Holland, 1994.*

*[9] C.C. Elgot. Decision problems of nite automata design and related arithmetics. Trans. Amer. Math. Soc., 98:21{52,* *1961.*

*[10] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In 22nd ACM Symposium on Principles of Programming Languages, pages 85{94, 1995.*

*[11] E. A. Emerson and K. S. Namjoshi. Automatic veri cation of parameterized synchronous systems. In 8th Conference on Computer Aided Veri cation, LNCS 1102, pages 87{98, 1996.*

*[12] K. Fisler and C. Girault. Modelling and Model Checking a Distributed Shared Memory Consistency Protocol. In ICATPN'98. Springer, 1998.*

*[13] S.M. German and A.P. Sistla. Reasoning about systems with many processes. Journal of the ACM, 39(3):675{735,* *1992.*

*[14] N. Halbwachs, F. Lagnier, and C. Ratel. An experience in proving regular networks of processes by modular model checking. Acta Informatica, 22(6/7), 1992.*

*[15] J.G. Henriksen, J. Jensen, M. J rgensen, N. Klarlund, B. Paige, T. Rauhe, and*

1. *Sandholm. Mona: Monadic Second-Order Logic in Practice. In TACAS '95, volume 1019 of LNCS. Springer, 1996.*

*[16] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying in nite-state systems. In S. Graf and M. Schwartzbach, editors, TACAS'00, volume 1785. Lecture Notes in Computer Science, 2000.*

*[17] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In O. Grumberg, editor, Proceedings of CAV '97, volume 1256 of LNCS, pages 424{435. Springer, 1997.*

*[18] R.P. Kurshan and K. McMillan. A structural induction theorem for processes. In ACM Symp. on Principles of Distributed Computing, Canada, pages 239{247, Edmonton, Alberta, 1989.*

*[19] D. Lesens, N. Halbwachs, and P. Raymond. Automatic veri cation of parameterized linear networks of processes. In POPL '97, Paris, 1997.*

*[20] K. Li and P. Hudak. Memory Coherence in Shared Virtual Memory Systems. ACM Trans. on Computer Systems, 7(4):321{359,* *1989.*

*[21] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the veri cation of concurrent systems. Formal Methods in System Design, 6(1), 1995.*

*[22] S. Owre, J. Rushby, N. Shankar, and F. von Henke. "formal veri cation for fault-tolerant architectures: Prolegomena to the design of PVS". IEEE Transactions on Software Engineering, 21(2):107{125,* *1995.*

*[23] Z. Stadler and O. Grumberg. Network grammars, communication behaviours and automatic veri cation. In Proc. Workshop on Automatic Veri cation Methods for Finite State Systems, Lecture Notes in Computer Science, pages 151{165, Grenoble, France, 1989. Springer Verlag.*

*[24] W. Thomas. Automata on in nite objects. In Handbook of Theoretical Computer Science, Volume B: Formal Methods and Semantics, pages 134{191. Elsevier Science Publishers B. V., 1990.*

*[25] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants (extended abstract). In Sifakis, editor, Workshop on Computer Aided Veri cation, LNCS 407, pages 68{80, 1989.*