*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) *14 pages*

*Verifying a Network Invariant for All* Con gurations of the Futurebus+ Cache Coherence Protocol

*Marcel Kyas*

*Institute for Computer Science and Applied Mathematics, Christian-Albrechts-Universitat, D-24105 Kiel, Germany,* [*mky@informatik.uni-kiel.de*](mailto:mky@informatik.uni-kiel.de)

*Abstract*

*In this paper we describe a network invariant for all con gurations of the Future- bus+ Cache Coherence Protocol. The network invariant was computed with PAX and veri ed by a model checker. Using this invariant we are able to prove a speci - cation of cache coherence correct for an arbitrary number of components on a single bus of the system. This speci cation includes a progress property not proven yet. We show how the result for the single bus system can be extended to tree-shaped systems. This is, as far as we know, the rst uniform proof of the protocol with multiple data-buses.*

# *1 Introduction*

*The automatic or semi-automatic veri cation of parameterised networks, i.e.,* of the family of systems P = fPi j i 2 !g, where each Pi is a network con- sisting of i processes, is an interesting but diÆcult task: In [2] it is shown, that the veri cation problem for parameterised networks is undecidable in general. Nevertheless, automatic and semi-automatic methods for the veri-

*cation of restricted classes of parameterised networks have been developed,*

*e.g., [21,29,16,23,1,7,5]. These methods have not been applied to a real-world* example, yet. We demonstrate, that the method of abstraction of WS1S tran- sition systems [3,4,5] can be applied to the Futurebus+ Cache Coherence Protocol [19].

*Considerable e ort has been invested into the veri cation of the IEEE* Futurebus+ protocol [11,6,20,12]. Either these e orts have not presented a uniform proof of correctness, i.e., a proof which establishes correctness for all instances of the parameterised network, or they have only veri ed a subset of the cache coherence speci cation.

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

*We base our proof method on the method described in [3,4,5]. It consists* of building a model of the parameterised network in the theory of weak sec- ond order logic of one successor (WS1S) [8,15,28]. The dynamic behaviour is described with a WS1S transition system. An abstraction relation is for- mulated in WS1S. Both serve as input to the tools PAX, which computes an

*nite-state abstract system from the model using MONA [18]. The resulting*

*abstract system is suitable as an input to a symbolic model checker [26,10,12],* which validates the speci cation. Our speci cation is given in linear time tem- poral logic (LTL) [25]. It can be checked using the model checker NuSMV [9] is used.

*In [3,4,5] the parallel composition is asynchronous, i.e., based on an in-* terleaving semantics. To handle architecture of the Futurebus+ protocol we extend the method with synchronous parallel composition.

*Using this method we prove the protocol correct with respect to a speci -* cation of cache coherence, which includes a progress property, which has not yet been proven in literature. Moreover, by using compositional reasoning, we prove the protocol correct for tree structures.

*Road Map*

*In the next section we introduce the notation and de nitions used in this arti-* cle. We recall the de nition of WS1S transition systems. Section 3 describes our extension of the proof method in [3,4,5] to synchronous parallel compo- sition. A short description of the Futurebus+ Cache Coherence Protocol is given in Sect. 4. Section 5 recalls the de nition of L-simulation and network invariants, summarises the results of [22] and describes the construction of the network invariant for the general case. We close this article with a short conclusion and comparison to related work.

# *2 Preliminaries*

*In this section we brie y recall the basic de nitions of WS1S [8,15,28] and* boolean transition systems [3,4].

*Terms of WS1S are build from the constant 0, rst-order variables, and* the successor function t:t + 1 applied to a term t. Second-order terms are either second-order variables, the constant ; denoting the empty set, or terms of the form X [ X0, X \ X0, or X n X0, where X and X0 denote second order terms. Atomic formulae are of the form b, t = t0, t < t0, or t 2 X, where b is a propositional variable, t and t0 are both terms, and X is a second-order term. WS1S-formulae are build from atomic formulae by applying the boolean connectives as well as quanti cation over rst and second-order variables. First-order monadic formulae are WS1S-formulae in which no second-order variables occur. WS1S-formulae are interpreted over models that assign nite subsets of ! (the natural numbers) to second-order variables and elements of

*! to rst-order variables. The interpretation is de ned in the usual way [28].* Given a WS1S formula f , [[f ]] denotes the set of models of f . The set of free variables is denoted by FV(f ). By FV2(f ) we denote the free second-order variables, by FV1(f ) we denote the set of free rst-order variables. One says that f is ( rst-order) closed, if FV(f ) = ; (FV1(f ) = ;). WS1S is a decidable formalism. Given a WS1S formula f we can construct a nite automaton, which is enumerating [[f ]]. However, in [27] it is shown that the space needed to compute the set of models of a WS1S formula is bounded from below by a stack of powers of two the size of the length of the formula. Despite this discouraging fact, a decision procedure has been implemented in the MONA tool [18].

*We de ne components of a WS1S transition systems in a sub-language* of WS1S called AF(n). For a rst-order variable n let AF(n) be the set of formulae produced by the grammar:

*f ::= x 2 B j :f j f ^ f j f \_ f j f ! f j 8nx:f j 9nx:f ;*

*where B is a second-order variable and x is a position variable. If m 2 ! then*

*m denotes the set of evaluations such that (n) = m, 0 (x) < m, and*

*(B) f0;::: ; m 1g. 8nx:f abbreviates the formula 8x:(0 x^x < n) ! f ,* and 9nx:f abbreviates the formula 9x:(0 x ^ x < n) ^ f . Similarly for second-order variables B we abbreviate the formula 8i:i 2 B ! f by 8Bi:f and 9i:i 2 B ^ f by 9Bi:f . The formulae in AF(n) are interpreted over

*evaluations 2 Sm2! m in the usual way.*

*De nition 2.1 A boolean transition system (BTS) S(i; n) = ( ; V; T ) pa-* rameterised by i and n consists of

*a nite set V of boolean array variables. A state of a BTS is a valuation of the variables of V . The set of all states is denoted by .*

*a satis able assertion 2 AF(n) with FV( ) V [ fig characterising the* set of initial states.

*a set T of transitions. Any transition 2 T can be represented by a predi-* cate (V; V 0) 2 AF(n) with FV( ) V [ V 0 [ fig, relating the unprimed

*variables in V to their primed successors in V 0. We require an idling tran-* sition I 2 T with I(V; V 0) = V (v $ v0).

*v2V*

*If (s; s0) j= (V; V 0), then s0 is a -successor of s, and we will write s !*

*s0.*

*A computation of a BTS S(i; n) = ( ; V; T ) is an in nite sequence such that:* Initiality: (0) j= .

*Consecution: For all i 2 ! we have (i) !S (i + 1) for some 2 T .*

# *3 Synchronous Monadic Parameterised Systems*

*In [3,4] a proof method is described for the abstraction of parameterised net-* works. This proof method is based on an asynchronous, i.e., interleaving

*based, parallel composition operation. In this section we extend thos proof* method to synchronous parallel composition. To this end, we de ne syn- chronous monadic parameterised systems.

*De nition 3.1 The synchronous monadic parameterised system (SMPS),*

*built from a BTS S(i; n), is the family P = fkm 1 S(i; n)[i=l; n=m] j m 2 ! g,*

*l=0 >0*

*where [ = ] is the substitution operation and k is the synchronous parallel com-*

*position.*

*A SMPS can be represented as a WS1S transition system (WS1S-TS).* De nition 3.2 A WS1S transition system S = ( ; V; T ) is de ned by:

*V is a nite set of second order variables, where each variable is interpreted*

*as a nite set of natural numbers.*

*The initial condition is given by a satis able assertion with FV( ) V .*

*T is a nite set of transitions, where each is represented by a WS1S* formula (V; V 0) with FV( ) V [ V 0.

*The computations of S are sequences of states and de ned as usual. A SMPS* is translated into a bisimilar WS1S-TS by the following translation procedure: Let P = f1;::: ; ng be a set of process indices from 1 to n. We use a translation function which replaces all occurrences of n by max(P )+ 1. The predicate

*part(P; B) = ( ^*

*B;B0 2B^B6=B0*

*B \ B0 = ;) ^ [ B = P*

*B2B*

*determines whether the set system B is a partition of the set P . The transla-* tion procedure is:

*De nition 3.3 Consider an SMPS P built from S(i; n) = ( ; V; T ). Then* the WS1S-TS ( ~ ; V~ ; T~ ) is de ned by:

*V~ = V [ fP g, where P 62 V .*

*Let ~ = 9n:P = f1;::: ; ng^ (VB2V B P ) ^ (8P m: ).*

*Let T~ = f ~ g, where ~ = 9 2T Y :part(P; fY j 2 T g) ^ (VB2V B*

*P ) ^ (P = P 0) ^ (8 2T i : V*

*2T*

*(i 2 Y ! [i=i 2 Xp])) and i2f1;:::;ngxi*

*abbreviates x1 : x2 : xn: for 2 f8; 9g.*

*The transition relation is given by just one formula. However, the space needed* to compute the set of models of a WS1S formula is bounded from below by a stack of powers of two the size of the length of the formula [27]. In our case study this resulted in a formula which is to large to be handled with our resources. This makes it necessary to keep the formula of the transition relation small. In [22] a method to remedy this situation is presented.

# *4 The Futurebus+ Cache Coherence Protocol*

*The intention of this section is to give an overview of how the IEEE Future-* bus+ Protocol [19] works. It de nes a cache to be a number of so-called cache

*next(cmd) := case*

*state=invalid: {none,read-shared,read-modified}; state=shared-unmodified: {none,invalidate,read-shared,*

*read-modified}; state=exclusive-unmodified: {none,copyback}; state=exclusive-modified: {none,copyback};*

*esac;*

*Fig. 1. SMV Code of a Cache-Line's Command Part*

*lines. Part of this standard is a cache-coherence protocol designed to work* in a hierarchically-structured multiple-bus system. The protocol maintains coherence on individual buses by having the caches snoop, or observe, all bus transactions. Coherence across buses are maintained using bus-bridges. Spe- cial agents, called cache and memory agents, at the ends of bridges represent remote caches and memories. In order to increase performance, the protocol uses so-called split-transactions. When a transaction is split, its completion is delayed and the bus is freed; at some later time, an explicit response is issued to complete the transaction. This facility makes it possible to service local requests while remote requests are being processed.

*The state of a cache line is characterised by the four attributes invalid,* shared-unmodi ed, exclusive-modi ed, and exclusive-unmodi ed. Figure 1 dis- plays the command part of the system in NuSMV syntax [9]. The following paragraphs give an informal overview over all transitions.

*A transaction is a sequence of events on the bus. Basically, it is a bus com-* mand, which is describing the nature of the transaction. The exact semantics of a bus command may be modi ed by one of the bus attributes IV, SR, TF or WT. Each processor has a local copy of those attributes, called iv, sr, tf or wt respectively. Usually, if a the local copy of such an attribute is set, it is also set on the bus by an or (\_) operation.

*Initially, any cache line is invalid. An invalid cache line must issue a* command or snarf a copy of the desired data to change its state (snar ng means reading the data from a transaction issued by another component). It is possible for a cache line to transition to any other state from the invalid state. A module that initiates a read-shared transaction and completes it successfully transitions to shared-unmodi ed state if TF is asserted and to exclusive-unmodi ed state, otherwise. It may assert tf during a read-shared, read-invalid or copy-back transaction to change its state to shared-unmodi ed. A module initiating a read-modi ed transaction that completes successfully will cause its cache line to change state to exclusive-modi ed.

*If a cache line is in shared-unmodi ed state, it may change its state to* invalid state without a bus transaction at any time. It is required to do so if it snoops a read-modi ed, write-invalid, or invalidate transaction. If it snoops a read-invalid or read-shared transaction and does not assert tf to snarf the data, it changes state to invalid, too. A module that initiates a invalidate

*transaction that completes successfully changes state to exclusive-modi ed. It* does not change state otherwise.

*Without using any bus transaction a module in exclusive-unmodi ed state* may change its state to any of the three others at any time. If it snoops a read-shared or read-invalid transaction during which it asserts tf it changes state to shared-unmodi ed and to invalid otherwise. It is required to do so if it snoops a read-modi ed, write-invalid or invalidate transaction.

*A processor may change the state of a cache line in exclusive-modi ed state* to invalid or shared-unmodi ed by initiating a copy-back transaction. It must assert tf to change state to shared-unmodi ed. If this module snoops a read- shared, read-invalid or read-modi ed transaction, it must assert iv to intervene and supply the data in place of the memory. It may keep a shared-unmodi ed copy if it asserts tf during a read-shared or read-invalid transaction. It al- ways changes the state of a cache line to invalid if it snoops a write-invalid transaction.

*For any command with WT asserted, a cache line shall not change its* state.

*Furthermore, modules may split transactions if access time of a module is* slow compared to bus access time. Cache and memory modules determine if they need to split a transaction by decoding the address and command for each cache-coherent read transaction that they snoop. If the module is responsible for that address and cannot respond immediately, it asserts sr.

*If a module splits a read-shared or read-invalid transaction, it must even-* tually respond with a shared-response transaction. This transaction may be snarfed by any cache. The memory must snarf data from this transaction. If no other cache signals that it has the cache line or that it is snar ng the data by asserting tf on a shared-response transaction, the line may be tagged exclusive-unmodi ed by the requester. Otherwise, it is shared.

*If a module splits a read-modi ed transaction, it must eventually respond* with a modi ed-response transaction. This transaction may not be snarfed. The module that originally generated a read-modi ed or invalidate transaction and subsequently received a modi ed-response changes its cache line state to exclusive-modi ed.

*Modules that split a transaction are required to assert wt if any other* module initiates a request transaction to the same cache line. This forces a limit of a single outstanding transaction per cache line. When a module initiating a transaction receives wait status, it has to wait until a shared- response or modi ed-response is observed by the cache line, and then it may repeat its request. In case of a shared-response transaction it may snarf the data and satisfy its request without a bus transaction.

*Specifying Cache Coherence.*

*We have based our speci cation of cache coherence on [11,12]. One speci ca-* tion was replaced by a progress property with an fairness assumption. To do this, the speci cation was rewritten in LTL [25].

*In [19] illegal attribute combinations are speci ed. Whenever such a com-* bination is observed, the system sets a bus-error. Even if the observed trans- action is legal, it may indicate an error. The absence of these error conditions is speci ed as (1).

*Next we want that at most one processor, indexed i, has got a writeable* cache line; this is expressed by (2).

*If any cache lines i and j have got a cache line in readable state, we require* that they agree on the data. If a cache line i is holding a readable copy, and the memory line is known to be unmodi ed, then both should agree on the data (see (3) and (4)).

*The fairness assumption states that each member on the bus will be* master of the bus in nitely often, a command is issued on the bus in nitely often, and that the wait attribute is not persistent, as expressed by (5). Under this assumption, we require that if a processor is waiting for a request, it will eventually receive an answer. Also, if a component is obligated to respond to a request, it will eventually ful l its obligation, as expressed by (6) and (7).

*(1) 2 :(bus-error \_ error )*

*(2) 8i:8j:i 6= j ! (2 pi:writeable ! :(pj:readable \_ pj:writeable))*

*(3) 8i:8j:i 6= j ! (2 pi:readable ^ pj:readable ! pi:data = pj:data)*

*(4) 8i:2 pi:readable ^ :m:memory-line-modi ed ! pi:data = m:data*

*(5) = (8d:d 2 D ! 2 3 d:master ) ^ (2 3 CMD 6= none) ^ (2 3 :WT )*

*(6) 8i: ! 2 (pi:requester 6= none ! 3 (pi:requester = none))*

*(7) 8i: ! 2 (pi:responder 6= none ! 3 (pi:responder = none))*

# *5 A Network Invariant for the Futurebus+*

*No one has yet presented a correct network invariant for the Futurebus+ Cache* Coherence Protocol yet. Let v denote that is a L-simulation [14] and *v*

*L L*

*that there exist an L-simulation. Then a network invariant is formally de ned*

*by:*

*De nition 5.1 Let (P; vL ) be a partially ordered set of processes, and k* monotone with respect to vL, i.e., for all P; Q 2 P with P vL Q and for all R 2 P we have P k R vL Q k R. Then we call a process I 2 P a network invariant, if it satis es P vL I and P k I vL I for all P 2 P.

*A network invariant is an abstraction for all members of a family [29] and by* this one has:

*Theorem 5.2 Let I be a network invariant for P and a speci cation. If*

*I j= , then for all P 2 P we have P j= .*

*By this theorem it is suÆcient to nd a network invariant that satis es a prop-* erty to show that all members of the family have this property. Unfortunately,

*nding a network invariant is a diÆcult task. One might ask whether such a* network invariant can always be found, or even whether one can compute one automatically. In fact, nding a network invariant is undecidable [29].

*5.1 A Network Invariant for the Futurebus+ Single-Bus Case*

*In this section we describe how we have built a network invariant for single-* bus con gurations of the Futurebus+ system. A very similar, but erroneous, network invariant can be found in [12].

*5.1.1 Building a parameterised model*

*The rst step in the construction of the network invariant was de ning a* parameterised model of the protocol as a WS1S transition system. In this particular case it was a simple task. We used the model described in [11] and translated it into a BTS (see Def. 2.1). Using the method described in Sect. 3 we have build a WS1S-TS from this description. The memory line and the bus itself are not considered in the model. They were added by hand to the WS1S-TS.

*Much e ort has been invested into minimising the WS1S-TS. The transi-* tion relation of the automata constructed from the WS1S description of the system may grow exponentially in the number of variables occuring in the system's description. Therefore, we minimised the number of second-order variables and the syntactic representation of the transition relation. This is described in detail in [22].

*5.1.2 Finding an abstraction relation*

*Our network invariant was automatically computed from the WS1S-TS using* the manually supplied abstraction relation:

*I $ (8i: pi:state = invalid) ^*

*SU $ (8i: pi:state = exclusive-modi ed ^* pi:state 6= exclusive-unmodi ed ) ^

*EM $ (9i: pi:state = exclusive-modi ed ^* 8j: i 6= j ! pj:state = invalid) ^

*EU $ (9i: pi:state = exclusive-unmodi ed ^* 8j: i 6= j ! pj:state = invalid) ^

*BAD $ :(I \_ SU \_ EM \_ EU ) \_ bus-error \_ error :*

*This is a \natural choice" for an abstraction relation. The idea is, that the* abstract process should be able to \mimic" every behaviour of the system it is abstracting. The same idea is also used in [12]. We introduce an abstract

*next(cmd) := case*

*state=invalid: {none,read-shared,read-modified}; state=shared-unmodified: {none,invalidate,read-shared,*

*read-modified}; state=exclusive-unmodified: {none,copyback,read-shared,*

*read-modified};*

*state=exclusive-modified: {none,copyback,read-shared,*

*read-modified};*

*esac;*

*Fig. 2. SMV Code of the Network-Invariant's Command Part*

*state BAD, which observes any violation of the safety properties de ned in* speci cations (1) and (2).

*The command part of the abstract system is given in Fig. 2. It di ers from* the original system only in the way the system issues the next command. By comparing Fig. 2 to Fig. 1, these changes become apparent:

*In exclusive-modi ed and exclusive-unmodi ed a read-modi ed and a read-* shared transaction may be initiated.

*If a cache line is in shared-unmodi ed state, it may issue a read-modi ed* and a read-shared command.

*The next step is to prove whether the system constructed this way is indeed* a network invariant. To do this we have to check whether the system abstract system and the abstraction relation satisfy Def. 5.1.

*The rst requirement (P v I) holds, if the abstraction relation implies*

*L*

*the identity relation, i.e., id , where is the set of states of S(0; 1). For*

*the second requirement (P k I v I) we can use the fact that the parallel*

*L*

*composition operation used here provides us with an intersection semantics* (see [22] for a proof of this statement), i.e., [[P k I]] = [[P ]] \[[I]] = [[P ]], because id *Finally we have to check, whether the parallel composition operation*

*is indeed monotonic with respect to our abstraction relation. Fortunately, this*

*is the case, as stated by the following Lemma:*

*Lemma 5.3 Let S1 = ( 1; V; T1 ), S2 = ( 2; V; T2), and S3 = ( 3 ; V; T3) be*

*BTS, the set of states of Si for i 2 f1; 2; 3g, an abstraction such that*

*id , S1 v S2, and S1 v S3 . Then S1 k S2 v S2 k S3 .*

*L L L*

*Proof. Because id the claim is equivalent to [[S1 ]] \ [[S2 ]] [[S2 ]] \ [[S3 ]].* By S1 v S2 we have [[S1 ]] \[[S2 ]] = [[S1 ]]. From [[S1 ]] [[S2 ]] and [[S1 ]] [[S3 ]]

*L*

*we have [[S1 ]] \ [[S1 ]] [[S2 ]] \ [[S3 ]], from which follows the claim.*

*Hence, for computing a network invariant in our framework, the abstraction* relation only needs to contain the identity relation between the parameterised system and the abstraction.

*Note, that the state BAD is not reached by the abstract system, i.e., the* graph immediately shows the validity of speci cations (1) and (2).

*next(cmd) := case*

*state=invalid: {none,copyback,read-shared,read-modified}; state=shared-unmodified: {none,invalidate,copyback}; state=exclusive-unmodified: {none,copyback,read-shared,*

*read-modified}; state=exclusive-modified: {none,copyback,read-shared,*

*read-modified};*

*esac;*

*Fig. 3. SMV Code of the Command Part of [12]*

*The reasons for these are: Recall how the next command may be issued by* a processor from Fig. 1. Assume the cache line is in exclusive-modi ed state. The processor must not, for example, issue a read-shared command. This can be done by a two processor con guration, which where the second processor is in invalid state.

*5.1.3 Comparison to [12].*

*In [12] a system is presented from which the authors claim that it is a network* invariant for the Futurebus+ Cache Coherence Protocol. Its command part is shown in Fig. 3. It is not a network invariant, because: A cache line in invalid state should not initiate a copyback transaction, because its data is, well, invalid. Such a transaction invalidates the content of the memory line. Moreover, a cache line in this model is not allowed to issue any read command if it is in shared-unmodi ed state. If one considers the system in shared-unmodi ed state and composes it with a cache line in invalid state, the system in [12] is no abstraction of this system, because it cannot mimic a read command issued by the invalid cache line.

*5.1.4 Veri cation*

*The network invariant in Fig. 2 satis es the speci cation of Sect. 4. The* proof was established with PAX and NuSMV. The computations involved took less than 2 Minutes. The speci cations (6) and (7) needed special treatment, because any read transaction may be split. This results in a computation invalidating this speci cation, where a transaction is always split. Using a proof rule from [4,5] we can prove that the system satis es 2 3 :SR. Hence we establish the speci cation for all single-bus con gurations.

*5.2 Generalising the Result*

*In this section we describe how to generalise the results from the linear case* to arbitrary network topologies. The construction of the network invariant for the general case is based upon the following two observations:

*(i) The memory board and the memory agent have a very similar behaviour.* If one considers the communication of the memory agent on the bus,

*one cannot distinguish it from a memory board (this is the intention of* its design). We were able to show MA vL M, where MA denotes the memory agent and M the memory board.

*(ii) A processor and a cache agent have a very similar behaviour, too. As* above, we have CA vL P , where CA is the cache agent and P the pro- cessor with its cache line.

*Using these abstractions and the fact that our parallel composition is* monotonic with respect to vL (cf. [22]) the network invariant in question turns out to be the network invariant for the single-bus case (rf. Sect. 5.1). This will be established in the following paragraphs.

*The rst step is to show, that the bus-bridge BB does not interfere with* normal operation. In our model of the protocol the bus bridge itself has an in- stant transmission time and only communicates the most necessary behaviour. The delays introduced by message propagation are simulated by having the agents non-deterministically split read-commands on their local bus. Then it easy to see, that BB vL B, where B is a data bus. We have already shown, that M k B k P vL I, where I is the network invariant. Using the mono- tonicity of k with respect to vL and the fact that vL is transitive, one has MA k BB k CA vL M k B k P vL I.

*The second step is to put these results together. A single-bus system* is composed of the data bus, a memory unit and an arbitrary number of processors. By the abstraction MA vL M one can replace the memory unit on a bus by an memory agent without changing its behaviour. Similarly any processor may be replaced by an cache agent. Hence the bus turned into a tree node. The main point is, that any of these substitutions have the same network invariant, because the memory-agent and the cache-agent are

*implementations of a memory or a cache, and each agent represents a complete* multi-bus system, representable by the network invariant, on the bus.

*Thus we have proven:*

*Proposition 5.4 I is a network invariant for any con guration of the Fu-* turebus+ Cache Coherence Protocol.

*This nishes the proof of the speci cation for all topologies of the Future-* bus+ Cache Coherence Protocol.

# *6 Conclusion*

*By extending the methods presented in [3,4,5] we are able to compute a* network invariant for the Futurebus+ Cache Coherence Protocol semi-auto- matically. This is, to the best of our knowledge, the rst correct network invariant. A new progress property not shown before was established (see Equations (5){(7)). This resulted in applying PAX to a large example. Us- ing compositional reasoning we have shown that the network invariant for the general case of the Futurebus+ Cache Coherence Protocol is the same as the

*network invariant for the protocol's single-bus case.*

*It turns out that the time needed to compute and verify the network in-* variant is far less the time needed to model-check a small sample con guration consisting of two processors. This makes the method presented here promising for the application to other large examples.

*Related Work.*

*A precise model of the Futurebus+ cache coherence protocol was designed by*

*E. Clarke et al. [11]. They used temporal logic model checking to show that* the protocol satis es a formal speci cation of cache coherence. They have veri ed a selection of examples and not a parameterised version. We have based our speci cation of cache coherence on theirs and extended it with a new progress property.

*Parameterised versions of the protocol were veri ed in [20,6]. Both e orts* only verify a subset of the speci cation given in [11].

*In [20] the states of a network are encoded as a regular language over an* alphabet of the states of its component processes and the transition relation is represented by a nite state transducer. This idea is improved in [1,7], where this method is called regular model checking. This method has been implemented in the Pen program [24]. We have based our method on similar ideas. For some systems a network invariant can be computed using an algo- rithm proposed by D. Lesens [23], where widening techniques [13] are used to compute a network invariant for linear parameterised networks. This idea is similar to ours. All those methods are semi-automatic or may fail to terminate.

*We have used PAX to compute a network invariant [21,29] for single-bus* con gurations of the Futurebus+ Cache Coherence Protocol. Such network invariants are described in [12,22]. The network invariant given in [12] is not correct. Though the abstraction relations are similar, the proof in [12] was done by hand. Our semi-automatic method increases the con dence in the established proof.

*In [16] it is shown that the veri cation problem for the linear Futurebus+* Cache Coherence Protocol is decidable. However, the veri cation with PAX is very fast. It only needed 2 Minutes to verify the system. Using a conven- tional model checker and 2 processors needed 36 hours to model check our speci cations.

# *References*

*[1] P. A. Abdulla, A. Bouajjani, B. Johnson, and M. Nilsson. Handling global conditions in parameterized system veri cation. In Halbwachs and Peled [17].*

*[2] K. R. Apt and D. C. Kozen. Limits for automatic veri cation of nite-state concurrent system. Information Processing Letters, 6(22), 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, Proc. TACAS '00, volume 1785 of LNCS. Springer, 2000.*

*[4] K. Baukus, Y. Lakhnech, and K. Stahl. Verifying universal properties of parameterized networks. In M. Joseph, editor, Proc. FTRTFT 2000, volume 1926 of LNCS. Springer, 2000.*

*[5] Kai Baukus, Yassine Lakhnech, and Karsten Stahl. Ver cation of parameterized protocols. Journal of Universal Computer Science, 7(2), 2001.*

*[6] S. Bensalem, Y. Lakhnech, and S. Owre. Computing abstractions of in nite state systems automatically and compositionally. In A. J. Hu and M. Y. Vardi, editors, Proc. CAV '98, volume 1427 of LNCS. Springer, 1998.*

*[7] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touilli. Regular model checking. In E. A. Emerson and A. P. Sistla, editors, Proc. CAV '00, volume 1855 of LNCS. Springer, 2000.*

*[8] J. Richard Buchi. Weak second-order arithmetic and nite automata. Zeitschrift fur mathematische Logik und Grundlagen der Mathematik, 6, 1960.*

*[9] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model veri er. In Halbwachs and Peled [17].*

*[10] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, Workshop on Logics of Programs, volume 131 of LNCS. Springer, 1981.*

*[11] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Veri cation of the Futurebus+ cache coherence protocol. In*

*L. Claesen, editor, Proc. 11th Int. Symp. Comp. Hardware Desc. Lang. App. North Holland, 1993.*

*[12] E. M. Clarke, O. Grumberg, and D. A. Peled. Model Checking. MIT Press, 1999.*

*[13] R. Cousot and P. Cousot. Abstract interpretation: A uni ed lattice model for static analysis of programs by construction or approximation of xpoints. In Proc. 4th ACM Symp. on Princ. of Prog. Lang., 1977.*

*[14] W.-P. de Roever and K. Engelhardt. Data Re nement: Model-Oriented Proof Methods and their Comparison. Cambridge University Press, 1998.*

*[15] C. C. Elgot. Decision problems of nite automata design and related arithmetics. Trans. Amererican Mathematical Society, 98, 1961.*

*[16] E. A. Emerson and K. S. Namjoshi. Automatic veri cation of parameterized synchronous systems. In R. Alur and T. A. Henzinger, editors, Proc. CAV '96, volume 1102 of LNCS. Springer, 1996.*

*[17] N. Halbwachs and D. Peled, editors. Proc. CAV '99, volume 1633 of LNCS. Springer, 1999.*

*[18] 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.*

*[19] IEEE standard for Futurebus+|logical protocol speci cation, 1992.*

*[20] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In O. Grumberg, editor, Proc. CAV '97, volume 1254 of LNCS. Springer, 1997.*

*[21] R. Kurshan and K. L. McMillan. A structural induction theorem for processes. In ACM Symp. on Princ. of Dist. Comp., 1989.*

*[22] M. Kyas. Veri kation parameterisierter Netzwerke durch Abstraktion. Master's thesis, Institut fur Informatik und Praktische Mathematik, Christian-Albrechts- Universitat, Kiel, 2000. In English, available at http://www.informatik. uni-kiel.de/~mky/publications/.*

*[23] D. Lesens. V eri cation et synth ese de syst emes r eactifs. PhD thesis, Institut National Polytechnique de Grenoble, September 1997.*

*[24] M. Nilsson. Analyzing parameterized distributed algorithms. Master's thesis, Dept. of Computer Systems, 1999.*

*[25] A. Pnueli. The temporal logic of programs. In Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, 1977.*

*[26] J. P. Queille and J. Sifakis. Speci cation and veri cation of concurrent systems in cesar. In M. Dezani-Ciancaglini and M. Montanari, editors, Proc. 5th Int. Symp. on Programming, volume 137 of LNCS. Springer, 1981.*

*[27] L. Stockmeyer. The complexity of decision problems in automata theory and logic. PhD thesis, Department of Electrical Engineering, M.I.T., Cambridge, M.A., 1974.*

*[28] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, Handbook of Formal Languages, volume III. Springer, 1997.*

*[29] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In J. Sifakis, editor, Proc. CAV '89, volume 407 of LNCS. Springer, 1989.*