Verification of a Parameterized Bus Arbitration Protocol *

E. Allen Emerson Kedar S. Namjoshi

Department of Computer Sciences,
The University of Texas at Austin, U.S.A.

Abstract.

Model Checking is well established as a verification technique for finite-state systems. Several important types of systems, such as protocols parameterized by the number of processes, are however inherently infinite-state, hence Model Checking cannot be applied directly to determine correctness of the system.

We present here a case study on the verification of such a parameterized protocol, the SAE-J1850 data transfer procotol. This is an standard in the automobile industry, where it is used to transmit data between various sensors and micro-controllers in an automobile. The protocol communicates data over a single-wire bus, and provides on-the-fly arbitration between competing transmissions. Our verification effort is interesting from many aspects : it proves correctness for arbitrary instances, is largely automated, and uses abstraction in an essential way. The abstractions used are exact, in the sense that a property is true of the parameterized protocol iff it is true of the finite-state abstraction.

A **Category A** submission. The contact author is Kedar S. Namjoshi. email : kedar@cs.utexas.edu; address : Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, U.S.A.

*This work was supported in part by NSF grant CCR 941-5496 and SRC Contract 97-DP-388. The authors may be reached at {emerson,kedar}@cs.utexas.edu.

1 Introduction

Model Checking [CE 81] (cf. [QS 82, CES 86]) is well established as a verification technique for finite-state systems. Many communication protocols, however, are parameterized by the number of processes, which induces an infinite family of (usually) finite-state instances. While Model Checking may be used to verify correctness of individual instances, this does not provide any guarantee that the entire family of instances is correct [1]. Thus it is an important research task to develop algorithms and semi-algorithmic procedures to verify parameterized systems.

The general problem is known to be undecidable [AK 86]; however, algorithms exist for specific types of systems (cf. [GS 92, EN 95, EN 96]), and semi-algorithmic procedures have been proposed to deal with general systems (cf. [CG 87, SG 89, KM 89, WL 89, PD 95, CGJ 95]). We present here a case study on the verification of an industrial standard parameterized protocol. The protocol is called the SAE-J1850 protocol [SAE 92], and is an automobile industry standard for transmitting data between various sensors and controllers in an automobile. The system consists of a single-wire bus, to which several controllers (units) are attached. Since the bus is a single wire, symbols **0** and **1** are transmitted by encoding them by both the length and the value of a bus pulse. For instance, a **0** may be sent with either a long high or a short low pulse. Several units may transmit concurrently; the protocol incorporates a *distributed, on-the-fly* arbitration mechanism which ensures that only the units transmitting the highest priority message succeed. Priority between messages (strings over $\{\mathbf{0}, \mathbf{1}\}$) is determined by lexicographic order, given that the symbol **0** has priority over the symbol **1**. The protocol is correct if it ensures that the arbitration mechanism functions correctly. We should note here that the protocol as described in [SAE 92] has other higher-level functionality, which we have not considered, in order to concentrate our attention on the core arbitration question.

The protocol is further complicated by the presence of arbitrary, but bounded delays in the units sensing changes in the global bus state. These delays have an electrical origin; they arise typically because of delays in the detection circuitry, and the presence of different bias voltages at the units. To accommodate these delays, "long" and "short" are actually time intervals, whose length is proportional to the maximum delay. Thus the protocol is parameterized both by the maximum delay, and by the number of units taking part in it. The verification of the protocol proceeds by two applications of abstraction, one for each parameter. The first abstraction theorem shows a *delay independence* property of the protocol : an instance of the protocol with n processes and maximum delay δ is correct iff the instance with n processes and maximum delay 2 is correct. Thus, correctness need be proved only for the family of instances with maximum delay 2. The second abstraction uses the algorithm in [EN 96] to handle the parameterization over the number of units in a *fully* automated manner; the algorithm constructs a *finite* "abstract graph", which represents the entire family of instances exactly, over which properties can be model-checked. In the [EN 96] paper, a simplified version of this protocol, without the complexity introduced by the delays, was verified. The modeling of the delay not only introduces complexity into the behavior of the units, but also introduces additional parameterization into the protocol, which is dealt with using the delay independence theorem.

The success of this effort leads us to believe that careful specification of the computational model underlying other protocols will expose constraints that can be utilized, as in this case, for developing decision procedures for large classes of protocols. In addition, this exposes a dire need for developing and popularizing notation for expressing such protocols. Remarkably, the SAE-J1850 document does not contain a succinct protocol description; the development of such a description was a major component of this project. The successful verification of the protocol using symbolic methods, despite the theoretical result on PSPACE-completeness of the procedure used [EN 96], is reason to believe that fully automated parameterized verification is feasible for reasonably sized protocols.

The rest of the paper is structured as follows: Section 2 describes the various components of the protocol in more detail. Section 3 discusses the abstractions used for handling the parameterization. In Section 4, we describe the implementation of the [EN 96] algorithm, and its application to this protocol. Section 5 concludes the paper and provides comparisons with related work.

[1] This situation is, ironically, similar to testing; verifying a few individual instances may help detect bugs, but can never prove their absence for the entire family.

2 Protocol Description

The SAE-J1850 protocol is a data transfer protocol over a single wire bus, which is intended to be used for communication between various sensors and controllers in an automobile. The restriction to a single wire bus reduces wiring complexity.

An instance of the system consists of several *units* connected to a single *bus*. The units communicate by broadcasting messages (sequences of symbols from the set $\{\mathbf{0}, \mathbf{1}\}$) over the bus. Units may transmit concurrently; arbitration takes place during transmission. The arbitration mechanism is defined in terms of priority among symbols; the symbol 0 has higher priority than 1. The priority order among symbols is extended to messages as lexicographic ordering. The key correctness property of this protocol is that the arbitration mechanism works as follows : whenever several units are sending messages concurrently, the message with the highest priority is placed on the bus.

As the bus is restricted to a single wire, symbols are encoded essentially by pulses of differing length. For instance, a **0** symbol is encoded with either a "long" high pulse, or by a "short" low pulse. The high and low states on the bus are referred to as *Dominant* and *Passive* respectively in the SAE-J1850 document [SAE 92], so we will use this terminology in the rest of the chapter. The state of the bus is an "or" of the bus states desired by the units. The protocol is further complicated by non-deterministic, though bounded delays in the units when sensing a bus change. This delay is caused either by bias voltages, or by delays in the detection circuitry. To account for these delays, "long" and "short" are not fixed numbers in the protocol, but are instead non-empty intervals, whose length is proportional to the maximum delay parameter, which we term Δ.

We will continue to use the symbolic names "long" and "short". There are four parameters associated with a symbolic length l : $Txmin(l)$, $Trmin(l)$, $Txmax(l)$, $Trmax(l)$. Their values are based on a nominal value $Tnom(l)$ and are given by the formulae : $Txmin(l) = Tnom(l) - \Delta/2$, $Txmax(l) = Tnom(l) + \Delta/2$, $Trmin(l) = Tnom(l) - 3\Delta/2$, $Trmax(l) = Tnom(l) + 3\Delta/2$. Thus, the interval $[Txmin(l), Txmax(l)]$ is contained in the $[Trmin(l), Trmax(l)]$. $Tnom(l)$ is itself proportional to Δ. $Tnom(Long) = 8 * \Delta$, and $Tnom(Short) = 4 * \Delta$. The values are given explicitly in the table below:

Length	*Trmin*	*Txmin*	*Txmax*	*Trmax*
Short	2.5Δ	3.5Δ	4.5Δ	5.5Δ
Long	6.5Δ	7.5Δ	8.5Δ	9.5Δ

Notice that the values increase in increments of Δ, and that the least *Long* value exceeds the largest *Short* value by Δ. The core of the protocol is the following procedure followed by each unit to transmit a symbol s with symbolic length l at a bus value of b (e.g., **0** as a *Long Passive* pulse). *localbus* is the bus value perceived by the unit (which may differ from the actual bus value during a bus transition), *request* is the bus value desired by the unit at the next cycle, and *counter* is the internal count of the number of cycles elapsed for this transmission. At the entry to this procedure, $localbus = b$ and $counter = 1$.

```
do
    counter ∈ [0, Trmin(l)) ⟶
        if
            localbus = b ⟶ request, counter := b, counter + 1
        []  localbus ≠ b ⟶counter := 1; raise FAILURE(* pulse too short *)
        fi
[]  counter ∈ [Trmin(l), Txmin(l)) ⟶
        if
            localbus ≠ b ⟶counter := 1; raise SUCCESS
        []  localbus = b ⟶ request, counter := b, counter + 1
        fi
[]  counter ∈ [Txmin(l), Txmax(l)) ⟶
        if
            localbus ≠ b ⟶counter := 1; raise SUCCESS
        []  localbus = b ⟶ request := ¬(b)
        fi
[]  counter ∈ [Txmax(l), Trmax(l)] ⟶
        if
            localbus ≠ b ⟶counter := 1; raise SUCCESS
        []  localbus = b ⟶ request, counter := Passive, counter + 1
        fi
[]  counter > Trmax(l) ⟶raise FAILURE (* pulse too long *)
od
```

Algorithm for transmitting a symbol.

2.1 Correctness Properties

The correctness property was stated informally in the protocol description as :

Whenever several units are transmitting messages concurrently, the message with the highest priority is the one placed on the bus.

This property can be stated precisely in *CTL* as follows: Consider n units connected to the bus, indexed by $i \in [1, n]$. Let $\mathcal{M}(k)$ denote the set of message strings (over $\{\mathbf{0}, \mathbf{1}\}$) of length k. For each i in $[1, n]$, let msg_i denote the message string from $\mathcal{M}(k)$ that is associated with unit i. Let $\max_i\{msg_i\}$ denote the maximum message, according to the lexicographic order $\preceq$ on messages. Let B denote the message that is transmitted on the bus (this may be defined as an auxiliary variable that records symbols as they are transmitted on the bus). Let tr_i be a boolean auxiliary variable that records if unit i is transmitting. The following *CTL* formula expresses the property above:

(C0) $(\forall m : m \in \mathcal{M}(k) : \mathsf{AG}((\exists i\; tr_i \wedge msg_i = m) \wedge (\forall i\; msg_i \preceq m) \wedge B = \epsilon \Rightarrow \mathsf{AF}(B = m)))$

This expression is of finite length for fixed k. Verification of this property for a fixed k requires adding state to each unit to store message contents, which makes the state space intractably large. To solve this problem, we modify the environment of the protocol so that the message sent by a unit is generated on the fly. At any state, let $sent_i$ denote the message sent by a unit. The modified correctness property is as follows :

(C1) $\mathsf{AG}((\exists i\; tr_i) \wedge B = \epsilon \wedge (\forall i : tr_i : sent_i = \epsilon) \Rightarrow \mathsf{AG}((\exists i\; tr_i \wedge B = sent_i) \wedge (\forall j\; sent_j \preceq B))) \wedge \neg(\exists v : \mathsf{AG}(B = v \wedge (\exists i : tr_i)))$

Informally, this property states that starting at any state where both the message on the bus and that at the units is empty, at any point of time the message on the bus is equal to the lexicographic maximum of the messages sent by the currently transmitting processes. Furthermore, the message on the bus B must increase as long as there is a transmitting process.

While the new environment is simpler, the statement of the property still involves several (unbounded) auxiliary variables. Instead of checking this property, which refers to the history of a computation, we check several properties that deal with the transmission of a single symbol. We show in lemma 1 that their conjunction implies (C1).

The proposition *insymbol* is true only at those states at time at least Δ from a bus state change with no changes in between. Let *exists0sender* be true iff there exists a process whose current symbol is $\mathbf{0}$; *exists1sender* is true iff there exists a process whose current symbol is $\mathbf{1}$.

(C2a) In any global state where symbol transmission is in progress, and there is a process sending $\mathbf{0}$, the next bus value is zero. In *CTL*, this is specified as

$\mathsf{AG}(insymbol \wedge exists0sender \Rightarrow \mathsf{A}(insymbol\ \mathsf{U}\ (value = \mathbf{0} \wedge \neg insymbol)))$

A similar property for the symbol $\mathbf{1}$ is :

(C2b) In any global state where symbol transmission is in progress, if there is a process sending $\mathbf{1}$ and there is no process sending $\mathbf{0}$, the next bus value is $\mathbf{1}$.

$\mathsf{AG}(insymbol \wedge \neg exists0sender \wedge exists1sender \Rightarrow \mathsf{A}(insymbol\ \mathsf{U}\ (value = \mathbf{1} \wedge \neg insymbol)))$

While the properties above are global properties, the following are properties of every unit, expressed in an indexed temporal logic (cf. [RS 85, BCG 89]):

(C2c) In any global state where symbol transmission is in progress, and unit i is transmitting $\mathbf{0}$, then its transmission inevitably succeeds, and unit i does not fail before the bus symbol is determined.

$\bigwedge_i \mathsf{AG}(insymbol \wedge tr_i \wedge symbol_i = \mathbf{0} \Rightarrow \mathsf{A}(insymbol\ \mathsf{U}\ [\neg insymbol\ \mathsf{U}\ (insymbol \wedge tr_i)]))$

(C2d) In any global state where symbol transmission is in progress, and there is a process sending $\mathbf{0}$, if unit i is transmitting $\mathbf{1}$, then its transmission fails before the bus symbol is determined.

$\bigwedge_i \mathsf{AG}(insymbol \wedge exists0sender \wedge tr_i \wedge symbol_i = \mathbf{1} \Rightarrow \mathsf{A}(insymbol\ \mathsf{U}\ (insymbol \wedge \neg tr_i)))$

(C2e) In any global state where symbol transmission is in progress, and there is no process sending 0, if unit i is transmitting $\mathbf{1}$, then its transmission inevitably succeeds, and unit i does not fail before the bus value is determined.

$\bigwedge_i \mathsf{AG}(insymbol \wedge \neg exists0sender \wedge tr_i \wedge symbol_i = \mathbf{1} \Rightarrow \mathsf{A}(insymbol\ \mathsf{U}\ [\neg insymbol\ \mathsf{U}\ (insymbol \wedge tr_i)]))$

Lemma 1 *Properties (C2a)-(C2e) imply Property (C1).*

Proof.

A *stable state* on an execution sequence is the state Δ time units after a bus value change. By the protocol definition, in this state every process perceives the new bus value. Thus, a stable state is the first state for which *insymbol* is true after a bus change.

Consider any reachable state where some process is transmitting and $B = \epsilon$. We show by induction on the number of stable bus states on any computation from that state that the following claim holds:

(IH) At the start of the kth stable bus state, the message on the bus is the empty sequence, if $k = 0$; otherwise, it is the lexicographic maximum of the messages sent by processes that were transmitting at the start of the $k-1$th stable bus state. All units that have sent the message on the bus are transmitting, and all units sending messages that are lexicographically strictly smaller have stopped transmission. At the kth stable state, there is at least one transmitting process.

Basis : $k = 0$. The message on the bus as well as the message sent by every unit are both the empty sequence, so the claim holds.

Inductive step : Assume that (IH) holds at the start of the kth stable bus state. By (IH), there is at least one unit transmitting at the kth stable bus state. There are two cases to consider:

Suppose that some unit attempts to transmit 0 at the kth stable state. By (C2a), the next symbol on the bus is 0. By (C2c), any unit transmitting $\mathbf{0}$ does not fail before the $(k+1)$th stable state. By (C2d), all units attempting to transmit $\mathbf{1}$ fail before the $(k+1)$th stable state.

Suppose that some unit attempts to transmit $\mathbf{1}$ at the kth stable state, and no unit attempts to transmit $\mathbf{0}$ at that state. By (IH), all units with a lower priority prefix among the first k symbols have failed. By

(C2b), the next bus symbol is **1** , and by (C2e), every unit transmitting **1** is still transmitting at the $(k+1)$st stable state.

In either case, the inductive hypothesis holds. Furthermore, the proof shows that as long as there is a transmitting process, the bus value changes at the next stable state, hence (C1) is true. □

3 Abstractions

The procotol described above is parameterized by the maximum delay parameter, Δ, as well as the number of processes, N, taking part in it. Let $P(N, \Delta)$ stand for the instance of the protocol with N processes and delay Δ. These parameters make the protocol infinite state, so that Model Checking cannot be applied to determine correctness. In order to apply Model Checking, we apply two abstractions that reduce the problem to an *equivalent* finite-state problem. The first abstraction demonstrates a *delay insensitivity* property of the protocol : for every N, $P(N, \Delta)$ is correct iff $P(N, 2)$ is correct. Hence, protocol correctness need be checked for only the set of instances with maximum delay 2. However, there are infinitely many such instances, so this is still a parameterized problem. This problem can be solved using the algorithm presented in [EN 96]. This algorithm constructs a *finite* "abstract graph", which encodes exactly the instances of the system. Model Checking the abstract graph is thus equivalent to checking the parameterized system. The experimental details are presented in the following section.

3.1 Delay Insensitivity

The timing parameters, as noted in the protocol description, are proportional to the maximum delay parameter Δ of the system. In addition, each test of the counter is of the form $counter \in \langle l * \Delta, r * \Delta \rangle$ (the angled brackets indicate either a open or a closed end to the interval), and each assignment to the counter is of the form $counter := choose\langle l * \Delta, r * \Delta \rangle$, which assigns to the counter a value from the interval. For such a system working with an underlying dense time, it is easy to show that if the intervals $\langle l * \Delta, r * \Delta \rangle$ are changed to $\langle l, r \rangle$ (dividing through by Δ), the resulting un-parameterized system has the same computations w.r.t. the states of the processes as the original one. This is so since global states with identical local states and clocks related by scaling with Δ, are bisimilar. This class of systems thus forms a decidable instance of parameterized real-time reasoning (cf. [AHV 93]). Since our model of the bus system is over integer time (each transition takes 1 time unit), we cannot use this result. However, the protocol satisfies additional properties that make a similar reduction possible. We show that any execution of $P(n, d)$ (d even and at least 2) can be simulated by an execution of $P(n, 2)$, in the sense that the sequence of symbols on the bus is the same. The following key lemma is needed for the proof of the theorem below :

Lemma 2 *Let σ be an execution of $P(n, d)$ (d even and at least 2). Let l be the length of the time interval between the ith and $(i+1)$st stable bus states in σ (i.e., either Long or Short). Then*

1. *Every process sending a symbol with a different length is aborted by the start of the $i+1$st stable state, and*
2. *Every process sending a symbol with the same length is live at the start of the $i+1$st stable state.* □

Theorem 1 *Let σ be an execution of $P(n, d)$ (d even and at least 2). There is an execution γ of $P(n, 2)$ such that the sequence of symbols on the bus is identical in σ and γ.*

Proof.

We construct γ inductively, such that for each i, γ_i is a sequence ending at the ith stable state, that matches the prefix of σ up to its ith stable state. Let γ_0 equal σ_0.

Suppose that γ_i has been constructed so that γ_i ends in the ith stable state on γ, the symbols on the bus in γ_i and in the subsequence of σ up to and including the ith stable state are identical, and the local states of corresponding processes in the ith stable states are the same except for, possibly, the *counter* values. The *counter* values, must however, satisfy the relationship : for any processes p, q, $counter_p \leq counter_q$ in the ith stable state in σ implies that $counter_p \leq counter_q$ in the ith stable state in γ.

Let s_i be the symbol sent on the bus in between the ith and $(i+1)$st stable states on σ. Let p be the process that determines the bus change. For a *Passive* to *Dominant* change, p is the first process to request a *Dominant* bus state, and for a *Dominant* to *Passive* change, p is the last process to request a *Passive* bus state. At each stable state, all processes begin transmission of their symbol with the same request, which is the same as the current bus value. Thus, the change by process p can occur only at $counter_p = Txmin(l)$, where l is the length that p sends its symbol at. $Txmin(l) = (a/2) * \Delta$, for some a.

The order of *counter* values is the same in the ith stable state in γ. Hence, in every execution starting at the ith stable state in γ, process p still is one of the processes that determine the bus change. As the change of bus state occurs at the same multiple of Δ, the length, and hence the symbol is the same. There exists a execution where after the bus change the *counter* values for processes not aborted are chosen in the order of *counter* values at the $(i+1)$th stable state of σ. From the previous lemma, the processes un-aborted at the $(i+1)$st stable states in γ and σ are the same; hence, the inductive assumption holds at the $(i+1)$st stable state. $\square$

We obtain the following theorem as a corollary:

Theorem 2 (Delay Insensitivity) *$P(n,d)$ is correct for every even d, $d \geq 2$, iff $P(n,2)$ is correct.*

Proof.

The direction from left to right follows by instantiating d with 2. For the direction from right to left, note that if $P(n,d)$ is incorrect for some d, then it contains a computation where the sequence of symbols on the bus is not the lexicographically greatest. By the previous theorem, this computation can be simulated by one in $P(n,2)$, so $P(n,2)$ would be incorrect. $\square$

Proof of Lemma 2:

Note that at a stable state, all processes have the same requested bus state, although they may be transmitting different symbols with differing lengths. In the interval between stable states, for any pair of processes p, q, $|counter_p - counter_q| \leq \Delta$.

(i) The length of the interval is *Long*. Let p be the process determining the new symbol. As the bus change occurs when p's *counter* value equals $Txmin(Long)$,

$Txmin(Long) - \Delta \leq counter_q \leq Txmin(Long) + \Delta$, for any process q, i.e., $6.5\Delta \leq counter_q \leq 8.5\Delta$.

If q sends a symbol by a short pulse, as $Trmax(Short) < 6.5\Delta$, q will have aborted by the time that the bus changes state. If q sends by a long pulse, its *counter* value remains in the interval $[Trmin(Long), Trmax(Long)]$ up to the next stable state, by which time the new bus state is perceived by q. Hence, every process sending a different length aborts, and every process sending a symbol with the same length is live at the next stable state.

(ii) The length of the interval is *Short*. Let p be the process determining the new symbol. As the bus change occurs when p's *counter* value equals $Txmin_{Short}$,

$Txmin(Short) - \Delta \leq counter_q \leq Txmin(Short) + \Delta$, for any process q, i.e., $2.5\Delta \leq counter_q \leq 4.5\Delta$.

If q sends by a long pulse, then as $Trmin(Long) = 6.5\Delta$, q will have aborted by the next stable state (which occurs in the interval $[3.5\Delta, 5.5\Delta]$). If q sends by a short pulse, its *counter* value remains in the interval $[Trmin(Short), Trmax(Short)]$ up to the next stable state, by which time the new bus state is perceived by q. Hence, every process sending a different length aborts, and every process sending a symbol with the same length is live at the next stable state.

It is easily seen from the protocol that once a bus value changes, it remains constant up to the next stable state. $\square$

3.2 Many-Process Verification

The delay insensitivity theorem (Theorem 2) shows that it is both necessary and sufficient to check every instance with delay 2 in order to check correctness for instances over all other delay values. While this eliminates consideration of the delay parameter, the reduced system is still infinite-state, as it is parameterized by the number of processes (units) taking part in the protocol.

Verification of this parameterized system can be carried out fully automatically using the algorithm described in [EN 96]. This algorithm is based on a synchronous *control-user* model, where the instances of the parameterized system consist of a fixed control process C, and many copies of a fixed user process U. The n-process instance can thus be described by $C \parallel U_1 \parallel \ldots \parallel U_n$, where $\parallel$ denotes synchronous composition. In the SAE-J1850 protocol, the control process models the behavior of the bus, while the user process models the behavior of a single unit, together with some machinery for modeling the delays in detecting bus value changes.

The algorithm of [EN 96] constructs, for such a control-user parameterized system, an "abstract graph", which is a finite-state abstraction of the entire *family* of instances. The states of the abstract graph record only the state of the control process, and whether there exists at least one user process, or no user process, in each user local state. The lemma below gives a way of checking safety properties of the family. Liveness properties may be checked in two ways : (a) As the abstract graph simulates every instance, if the liveness property holds of the abstract graph, then it holds of the family, (b) An algorithm is provided in [EN 96] for exactly determining whether the liveness property holds of every instance.

Lemma 3 *[EN 96] The abstract graph simulates every instance of the family. Every finite path in the abstract graph corresponds to a finite computation of some instance.*

The paper also shows how to check properties of the form $\bigwedge_i \mathsf{A}g(i)$ by reducing them, using symmetry arguments (cf. [ES 93, CFJ 93]) to checking a property $\mathsf{A}g(0)$ of the control process in a modified control-user system, which has the same user process, but has $C' = C \parallel U$ as the new control process.

4 Implementation Details

The behavior of the bus and the units as specified in the protocol is coded as a SMV [McM 92] program. The transition relation of the abstract graph is generated automatically by a program which takes the specification of control and user processes (in C), and generates SMV code for the abstract graph transitions. This is done by enumerating the reachable local states for a single user process, then generating each transition of the abstract graph by inspection of the local transitions in the unit. States of the abstract graph are represented by subsets of the local user state space. Each subset indicates the presence of at least one user process in that local state, as discussed in the previous section. Thus, for a local user transition $s \longrightarrow t$, the corresponding abstract graph transition adds t as a member of the next abstract state, given that s is a member of the current one.

For the parameterized family checked with maximum delay 2, each unit has 254 reachable states; thus, the number of boolean variables needed to encode an abstract state is also 254 (subsets are encoded as a boolean membership vector). The correctness properties C2(a) - C2(e) were checked together on the abstract graph. Since some of these properties are liveness properties, they were checked on the abstract graph, using the fact that it simulates every instance. Every property succeeds on the abstract graph, so that can infer that properties C2(a) - C2(e) hold of the parameterized system with delay 2, which by Theorem 2 implies that they hold of the completely parameterized system. By Lemma 1, this implies that the desired correctness property, (C1), holds of the completely parameterized system. We did not have to invoke the potentially expensive but exact method for checking liveness properties.

These checks take about 8 MB and 35 seconds on an Intel Pentium 133 with 32 MB of main memory. Conjunctive partitioning of the transition relation and pre-computation of the reachable states (the strongest invariant) is used. 24 iterations are needed to compute the reachable state space. Incidentally, checking a 15 process instance takes roughly the same amount of time but less space.

5 Conclusions and Related Work

Verification of parameterized systems is often done by hand, or with the guidance of a theorem prover (cf. [MC 88, MP 94, HS 96]). Several methods have been proposed that, to various degrees, automate this verification process. Methods based on manual construction of a process invariant are proposed in [CG 87, SG 89, KM 89, WL 89, LSY 94], and have been applied for the verification of the Gigamax cache consistency protocol in [McM 92]. These constructions have been partially automated in [RS 93, CGJ 95] (cf. [V 93, PD 95, ID 96]); however, as the general problem is undecidable [AK 86], it is not in general possible to obtain a finite-state process invariant. For classes of parameterized systems obeying certain constraints, [GS 92, EN 95, EN 96] give algorithms (i.e., decision procedures) for model checking the parameterized system. These papers demonstrate the methods on simple verification examples; we believe that our case study is one of the few examples of verification of a large and complex parameterized protocol. It is likely that the delay insensitivity theorem is an instance of a general theorem for such types of systems; given such a theorem, the verification of this protocol could be indeed fully automated.

We believe that careful specification of the computational model underlying other protocols will expose constraints that can be utilized, as in this case, for developing decision procedures for large classes of protocols. There is also a need for developing and popularizing notations for expressing such protocols. Remarkably, in the SAE-J1850 document (over 100 pages), there is no succinct protocol description; the description given in Section 2 had to be culled from the entire text. The successful verification of the protocol, despite the theoretical result on PSPACE-completeness of the procedure [EN 96], is reason to believe that fully automated parameterized verification is feasible for reasonably sized protocols.

Acknowledgements We would like to thank Carl Pixley of Motorola for suggesting that we undertake this protocol verification, and for providing us with the SAE documents.

References

[AHV 93] Alur, R., Henzinger, T., Vardi, M. Parametric Real-Time Reasoning, STOC 1993.

[AK 86] Apt, K., Kozen, D. Limits for automatic verification of finite-state concurrent systems. IPL 15, pp. 307-309.

[BCG 89] Browne, M. C., Clarke, E. M., Grumberg, O. Reasoning about Networks with Many Identical Finite State Processes, Information and Computation, vol. 81, no. 1, pp. 13–31, April 1989.

[CE 81] Clarke, E. M., Emerson, E. A. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. in *Workshop on Logics of Programs*, Springer-Verlag LNCS 131.

[CES 86] Clarke, E. M., Emerson, E. A., and Sistla, A. P., Automatic Verification of Finite-State Concurrent Systems using Temporal Logic, *ACM Trans. Prog Lang and Sys.*, vol. 8, no. 2, pp. 244-263, April 1986.

[CFJ 93] Clarke, E. M., Filkorn, T., Jha, S. Exploiting Symmetry in Temporal Logic Model Checking, 5th CAV, Springer-Verlag LNCS 697.

[CG 87] Clarke, E. M., Grumberg, O. Avoiding the State Explosion Problem in Temporal Logic Model Checking Algorithms, PODC 1987.

[CGJ 95] Clarke, E.M., Grumberg, O., Jha, S. Verifying Parameterized Networks using Abstraction and Regular Languages. CONCUR 95.

[Em 90] Emerson, E. A. Temporal and Modal Logic, in Handbook of Theoretical Computer Science, (J. van Leeuwen, ed.), Elsevier/North-Holland, 1991.

[EN 95] Emerson, E. A., Namjoshi, K. S. Reasoning about Rings, POPL 1995.

[EN 96] Emerson, E. A., Namjoshi, K. S. Automatic Verification of Parameterized Synchronous Systems, CAV 1996.

[ES 93] Emerson, E. A., Sistla, A. P. Symmetry and Model Checking, 5th CAV, Springer-Verlag LNCS 697.

[GS 92] German, S. M., Sistla, A. P. Reasoning about Systems with Many Processes. J.ACM, Vol. 39, Number 3, July 1992.

[HS 96] Havelund, K., Shankar, N. Experiments in Theorem Proving and Model Checking for Protocol Verification, FME, 1996.

[ID 96] Ip, N., Dill, D. Verifying systems with replicated components in Murϕ, CAV 1996.

[KM 89] Kurshan, R. P., McMillan, K. A Structural Induction Theorem for Processes, PODC 1989.

[LSY 94] Li, J., Suzuki, I., Yamashita, M. A New Structural Induction Theorem for Rings of Temporal Petri Nets. IEEE Trans. Soft. Engg, vol. 20, No. 2, February 1994.

[Lu 84] Lubachevsky, B. An Approach to Automating the Verification of Compact Parallel Coordination Programs I. *Acta Informatica* 21, 1984.

[McM 92] McMillan, K., Symbolic Model Checking: An Approach to the State Explosion Problem, Ph.D. Thesis, Carnegie-Mellon University, 1992.

[MC 88] Misra, J., Chandy, K. M. *Parallel Program Design : A Foundation*, Addison-Wesley Publishers, 1988.

[MP 94] Manna, Z., Pnueli, A. Verification of Parameterized Programs. In Specification and Validation Methods (E. Borger, ed.), Oxford University Press, pp. 167-230, 1994.

[Pn 77] Pnueli, A. The Temporal Logic of Programs. FOCS 1977.

[PD 95] Pong, F., Dubois, M. A New Approach for the Verification of Cache Coherence Protocols. *IEEE Transactions on Parallel and Distributed Systems*, August 1995.

[QS 82] Queille, J.P., J. Sifakis, Specification and Verification of Concurrent Systems in CESAR, *Proc. of the 5th International Symposium on Programming*, LNCS#137, Springer–Verlag, pp. 337–350, April 1982.

[RS 85] Reif, J., Sistla, A. P. A multiprocess network logic with temporal and spatial modalities. *JCSS* 30(1), 1985.

[RS 93] Rho, J. K., Somenzi, F. Automatic Generation of Network Invariants for the Verification of Iterative Sequential Systems. CAV 1993, LNCS 697.

[SAE 92] SAE J1850 Class B data communication network interface. Society of Automotive Engineers, Inc., 1992.

[SG 89] Shtadler, Z., Grumberg, O. Network Grammars, Communication Behaviours and Automatic Verification. In J.Sifakis (ed), *Automatic Verification Methods for Finite State Systems*, Springer-Verlag, LNCS 407.

[VW 86] Vardi, M., Wolper, P. An Automata-theoretic Approach to Automatic Program Verification, Proc. IEEE LICS, pp. 332-344, 1986.

[V 93] Vernier, I. Specification and Verification of Parameterized Parallel Programs. Proc. 8th International Symposium on Computer and Information Sciences, Istanbul, Turkey, pp. 622-625.

[WL 89] Wolper, P., Lovinfosse, V. Verifying Properties of Large Sets of Processes with Network Invariants. In J.Sifakis (ed), *Automatic Verification Methods for Finite State Systems*, Springer-Verlag, LNCS 407.