Verification of a Parameterized Bus Arbitration Protocol *

E. Allen Emerson and 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. Many protocols, while composed of finite-state processes, are parameterized by the number of such processes, hence Model Checking cannot be applied directly to determine correctness of the inherently infinite-state parameterized system. We present a case study on the verification of such a parameterized protocol, the SAE-J1850 data transfer procotol. This is a 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 if and only if it is true of the finite-state abstraction.

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 is often used to verify correctness of individual instances, this does not provide any guarantee of correctness for the entire family [1]. Thus an important research task is 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 many semi-algorithmic procedures have been proposed (cf. [CG 87], [SG 89], [KM 89], [WL 89], [PD 95],[CGJ 95]).

We present a case study on the verification of an parameterized industrial standard protocol. The protocol is called the SAE-J1850 protocol [SAE 92], and

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

it 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 while detecting a change in the bus state. These delays have an electrical origin; they arise from 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 units and maximum delay Δ is correct iff the instance with n units 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. A simple version of this protocol, without the complexity introduced by the delays, was verified in [EN 96]. 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 by 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 other classes of parameterized protocols. It also 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 parameterizations. 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.

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 parameterized system consists of several *units* connected to a single *bus*. The operation of the protocol can be described at the "interface" and "implementation" levels.

At the interface level, 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 by 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.

At the implementation level, since the bus is a single wire, symbols are encoded by pulses of differing length and the bus value during the pulse. For instance, the **0** symbol is encoded by 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 paper. The state of the bus is an "or" of the bus states desired by the units. The protocol is further complicated by nondeterministic, but bounded delays in the units while detecting a change in bus value. 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, 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 : $Txmin(l) = Tnom(l) - \Delta/2$, $Txmax(l) = Tnom(l) + \Delta/2$, $Trmin(l) = Tnom(l) - 3\Delta/2$, $Trmax(l) = Tnom(l) + 3\Delta/2$. $Tnom(l)$ is itself proportional to Δ. $Tnom(Long) = 8 * \Delta$, and $Tnom(Short) = 4 * \Delta$. The values are given explicitly in the table below:

Note that the interval $[Txmin(l), Txmax(l)]$ is properly contained in the interval $[Trmin(l), Trmax(l)]$, 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

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

Fig. 1. Interval Lengths

unit to transmit a symbol with symbolic length l at a bus value of b (e.g., **0** as a *Short*, *Passive* pulse). At the entry to this procedure, $request = b$, $localbus = b$, and $counter = 1$.

```
var localbus (* the bus value perceived by the unit *)
var request (* the bus value desired by the unit at the next cycle *)
var counter (* the number of cycles elapsed for this transmission *)

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

Fig.2. Algorithm to transmit a symbol with length l and bus value b.

Informally, the procedure above attempts to maintain the bus at value b for $Txmin(l)$ time units. If this attempt succeeds, then it attempts to change the bus value to $\neg b$ within $Txmax(l)$ time units so as to terminate the pulse. If that fails, then the procedure switches to a *Passive* request and waits for some other unit to change the bus value. As the names indicate, $[Trmin(l), Trmax(l)]$ is the interval for successful "reception" of the symbol while $Txmin(l)$ and $Txmax(l)$

are the time bounds for attempting "transmission" of the symbol. **0** is encoded as either a *Short Passive* pulse or as a *Long Dominant* pulse, while **1** is encoded by the other two combinations. The asymmetry between *Passive* and *Dominant* is used to enforce the priority order $\mathbf{1} \preceq \mathbf{0}$.

2.1 Correctness Properties

The correctness property is stated informally in the protocol document [SAE 92] 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, $(i \in [1,n])$. 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 fixed message string that is associated with unit i. 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. Let max be the function that determines the maximum message of a set of messages, according to the lexicographic priority $\preceq$ on messages. If the set is empty, max has value ϵ, the empty string. The following *CTL* formula expresses the property above:

(C0) $(\forall m : m \in \mathcal{M}(k) : \mathsf{AG}(maxT = m \wedge B = \epsilon \Rightarrow \mathsf{A}(B \preceq m \,\mathsf{U}\, B = m)))$, where $maxT = max\{i : i \in [1,n] \wedge tr_i : msg_i\}$

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) $\mathsf{AG}(maxS = \epsilon \wedge B = \epsilon \Rightarrow \mathsf{AG}(B = maxS))$, where $maxS = max\{i : i \in [1,n] \wedge tr_i : sent_i\}$

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 units. This implies that B must increase (lexicographically) as long as there is a transmitting unit.

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 statement of these properties requires some auxiliary propositions : *insym* holds at states where $B = \epsilon$, or the state is at least Δ time units from the last bus state change; *E0sender* holds iff there is a transmitting unit with current symbol **0**; *E1sender* holds iff there is a transmitting unit with current symbol **1**.

Let $before(x) \equiv \mathsf{A}(insym \,\mathsf{U}\, (insym \wedge x))$, $at(x) \equiv \mathsf{A}(insym \,\mathsf{U}\, (\neg insym \wedge x))$, and $after(x) \equiv \mathsf{A}(insym \,\mathsf{U}\, (\neg insym \,\mathsf{U}\, (insym \wedge x)))$. Informally, $before(x)$ holds

iff x holds before every next bus change, $at(x)$ holds iff x holds at the following bus change, and $after(x)$ holds iff x holds just after the bus change is complete.

A *stable state* on an execution sequence is one where $B = \epsilon$, or the state is at Δ time units after the last bus value change. By the protocol definition, in this state every unit perceives the new bus value. A stable state is the first state for which *insym* is true after a bus change.

(C2a) In any global state where symbol transmission is in progress, and there is a unit sending **0**, the next bus value is **0**. In *CTL*, this is specified as

$\mathsf{AG}(insym \wedge E0sender \Rightarrow at(value = \mathbf{0}))$

(C2b) In any global state where symbol transmission is in progress, if there is a unit sending **1** and no unit sending **0**, the next bus value is **1**.

$\mathsf{AG}(insym \wedge \neg E0sender \wedge E1sender \Rightarrow at(value = \mathbf{1}))$

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, every unit transmitting **0** succeeds and continues to transmit until the next *insym* state.

$\bigwedge_i \mathsf{AG}(insym \wedge tr_i \wedge (sym_i = \mathbf{0}) \Rightarrow after(tr_i))$

(C2d) In any global state where symbol transmission is in progress, and there is a unit sending **0**, every unit transmitting **1** fails before the bus symbol is determined.

$\bigwedge_i \mathsf{AG}(insym \wedge E0sender \wedge tr_i \wedge (sym_i = \mathbf{1}) \Rightarrow before(\neg tr_i))$

(C2e) In any global state where symbol transmission is in progress and there is no unit sending **0**, every unit transmitting **1** succeeds and continues to transmit until the next *insym* state.

$\bigwedge_i \mathsf{AG}(insym \wedge \neg E0sender \wedge tr_i \wedge (sym_i = \mathbf{1}) \Rightarrow after(tr_i))$

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

Proof. We show by induction on the number of stable states on any computation from a state with $B = \epsilon$ and $maxS = \epsilon$ (the 0th stable state) that the following property holds:

(IH) At the the kth stable state, B is the maximum of the messages sent by units that were transmitting at the start of previous stable bus state if $k > 0$, otherwise it is ϵ. Every transmitting unit has sent B.

Basis : $k = 0$. The message on the bus as well as the message at every transmitting unit are both ϵ, so the claim holds.

Inductive step : Assume that (IH) holds at the kth stable state. If some unit transmits **0** at this state, by (C2a) the next symbol on the bus is **0**. By (C2c), any unit transmitting **0** is transmitting at the next stable state. By (C2d), all units transmitting **1** fail before the next stable state.

If some unit transmits **1**at this state and no unit transmits **0**, then by (C2b), the next bus symbol is **1**, and by (C2e) every unit transmitting **1** is still transmitting at the next stable state. By (IH), at the kth stable state, all units transmit the lexicographic maximum among the sent messages, hence, at the next stable state, the value of B is still the maximum among the messages sent. In either case, the inductive hypothesis holds. □

3 Abstractions

The procotol as described is parameterized by both the maximum delay parameter Δ, and the number of units N. Let $P(N, \Delta)$ stand for the instance of the protocol with N units and delay Δ. This parameterization makes the protocol infinite-state, hence Model Checking cannot be applied directly to determine its correctness. We apply two abstractions that reduce the protocol to an *equivalent* finite-state system. 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 only for the set of instances with maximum delay 2. However, this is still a parameterized, infinite-state protocol. This parameterization can be handled with the algorithm presented in [EN 96]. This algorithm abstracts away the number of units, constructing a *finite* "abstract graph" which encodes all instances of the system. Model Checking the abstract graph created by this unit is thus equivalent to checking the doubly parameterized SAE-J1850 protocol. Experimental details are presented in the following section.

3.1 Delay Insensitivity

As noted in the protocol description, the timing parameters are proportional to the parameter Δ. In an underlying dense time model, each test of a clock variable x is of the form $x \in \langle l * \Delta, r * \Delta \rangle$ (the angled brackets indicate either a open or a closed end to the interval), and each reset of x is of the form $x := \textbf{choose}\langle l * \Delta, r * \Delta \rangle$, which assigns to x a nondeterministically chosen value from the interval. It is then straightforward 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 unparameterized system has the same computations w.r.t. the non-clock variables as the original one. This is so since global states with identical non-clock values 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. The protocol, however, 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.

Lemma 2. *Let σ be an execution of $P(n, d)$ (d even and at least 2). Let l be the symbolic length of the time interval between successive stable bus states in σ. Then*

1. *Every unit sending a symbol with a different length is aborted by the start of the next stable state, and*
2. *Every unit sending a symbol with the same length is transmitting at the start of the next stable state.* □

Theorem 1. *Let σ be an execution of $P(n,d)$ (d even and at least 2) from a stable state. 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. For each i, γ_i ends in the ith stable state, 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 units in the ith stable states are the same except for, possibly, the *counter* values. The *counter* values, must however, satisfy the relationship : for any pair of units 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 γ_0 equal σ_0. Let p be the unit that determines the bus change that results in the $(i+1)$st stable state. For a *Passive* to *Dominant* change, p is the first unit to request a *Dominant* bus state, and for a *Dominant* to *Passive* change, p is the last unit to request a *Passive* bus state. At each stable state, all units begin transmission of their symbol with request identical to the current bus value. Thus, the change by unit 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. As the *counter* value in each unit does not decrease until a bus change or a termination of transmission, in every execution starting at the ith stable state in γ, unit p still is one of the units that determine the bus change. As the change of bus state occurs at the same multiple of Δ, the symbolic length, and hence the symbol on the bus is the same. From the previous Lemma, the units un-aborted at the $(i+1)$st stable states in γ and σ are the same. There exists a execution where within Δ units after the bus change, *counter* values for unaborted units are chosen in the order of *counter* values at the $(i+1)$th stable state of σ. Hence, the inductive hypothesis holds. □

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 maximum of the sent messages. By the previous theorem, this computation can be simulated by one in $P(n,2)$, so $P(n,2)$ is incorrect. □

Proof of Lemma 2:

Note that at a stable state, all units 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 units p, q, $|counter_p - counter_q| \leq \Delta$.

(i) The length of the interval is *Long*. Let p be the unit 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 unit 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 aborts 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.

(ii) The length of the interval is *Short*. Let p be the unit 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 unit 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 aborts 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 unit sending a different length aborts, and every unit sending a symbol with the same length is live at the next stable state. □

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 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 a finite-state "abstract graph" for such a control-user parameterized system which is an abstraction of the entire *family* of instances. The states of the abstract graph record only the state of the control process, and for each local user state, whether there exists at least one user process in that 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 describing the *transition relation* of the abstract graph. 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 $s \rightarrow t$, the corresponding abstract graph transition adds t as a member of a abstract state following one that has s as a member.

For the singly parameterized system with $\Delta = 2$, 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 we 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 unit 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 thank Dr. Carl Pixley for suggesting the verification of the SAE-J1850 protocol and for helpful remarks.

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 Syst.*, 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*, 1995.

[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.

[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, *LICS*, 1986.

[V 93] Vernier, I. Specification and Verification of Parameterized Parallel Programs. *Proc. 8th Intl. Symp. 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.