Verification of Embedded Systems Using Synchronous Observers

Martin Westhead[1] and Simin Nadjm-Tehrani[2]

[1] martinwe@aifh.ed.ac.uk, Dept. of Artificial Intelligence, University of Edinburgh, 5 Forrest Hill,Edinburgh, U.K.
[2] simin@ida.liu.se, Dept. of Computer Information Science, Linköping University S-581 83 Linköping, Sweden

Abstract. This paper is a study of observer-based proof techniques applied to the verification of a model of a real world embedded system, an aircraft landing gear. We present a formal description of these techniques (taken from [5]) and look at three ways of applying them, comparing verification of the composed system with two approaches to decompositional verification. The example illustrates that due to the tight interaction in a plant-controller setting there is often little to be gained by adopting a decompositional approach to verification. Nonetheless, two reasons are presented for separation between the controller and its environment at the modelling stage. Hence the result of the study is that in cases similar to this one, it is most expedient to prove system properties using the composed model derived from individual parts.

1 Introduction

A major application area for automatic verification is embedded systems where software controllers interact with a physical environment. Verifying safety and timeliness properties for such systems has recently attracted a lot of attention in the context of hybrid systems – where models with both continuous and discrete elements are subject to study[4]. In this paper we explore an alternative methodology, namely analysis of real-time control programs in combination with discrete abstractions of their environment (the plant).

As a formalism for this investigation we have chosen an abstraction of a synchronous program, a synchronous Input/Output (I/O) machine, as defined by Halbwachs et.al. [5]. This formalism is typically used to provide the formal semantics for programming languages such as ESTEREL and for defining the criteria for accepting well-behaved programs. Here we use the formalism for modelling embedded systems; this gives us the possibility of using the associated verification technology in our case study. This approach has obvious benefits. First, we use the ESTEREL development environment and verification techniques for proving properties which can not be proved using the model of the control program alone. These are properties which depend on the tight interaction between the plant and the controller. Secondly, once the composed model of the embedded system has been proved to have its desired properties the conversion of the real-time program to C code (or a circuit) is a matter of routine translation provided

within the ESTEREL environment. Thus, existing and well-understood verification techniques for discrete models can be applied in a new setting. The price to pay is the derivation of explicit models for the environment. In this paper we argue that this is a viable undertaking.

The plant model we use is based on the physically grounded models derived elsewhere[10]. It can also be seen as the discrete structure extracted from a hybrid model based on Delta IO machines currently under development [12]. The verification technique we employ is the observer-based method given by Halbwachs et.al. [5], where a safety property of an I/O machine M is defined in terms of another machine called a synchronous observer. The observer watches the inputs and outputs of M, and if they ever violate the safety property it emits an alarm signal. A restriction of the method is that only safety properties are expressible.

The observer-based technique can also be applied in a decompositional[3] way which has been suggested elsewhere as a means for reducing complexity of proofs [1, 7, 3, 2]. Supposing we have a system of two components, and some overall property we wish to prove that the system satisfies. Instead of composing the system and verifying the property directly, we wish to carry out a smaller proof step involving an intermediate property on each of the components. Unfortunately it is not always clear how to obtain this intermediate property nor for that matter whether these new proofs will be easier. In this paper we study this problem in the context of a an aircraft landing gear controller adopted from a real-world application[10]. The method is interesting to study since for our class of embedded systems there is an obvious (plant-controller) breakdown of the components. Thus, we study whether this particular decomposition pays off during verification.

The example illustrates the use of an explicit model of the environment for proving safety properties which depend on the correct interaction between the plant and the controller. The plant model can be used in a one shot compositional verification of the embedded system, or alternatively for deriving the weakest observer leading to proofs of safety property in a decompositional framework. In the paper we explain, again in the context of this example, why both of these approaches are more appropriate than decompositional verification using ad hoc intermediate properties.

In the next section we present the formal definition of an observer with other definitions required for the rest of the paper. Section 3 describes a model of the aircraft landing gear in terms of synchronous I/O machines, and compares different proofs of a safety property. Finally in section 4 we return to the theory, and consider how the automatic synthesis of an intermediate property can be carried out.

Earlier applications of the automata approach within control systems can be found in [11]. A more recent use of the observer based methodology within

[3] Note that the term 'compositional verification' is sometimes used for the approach we denote by decompositional in this paper. We shall use the term compositional verification when we refer to a one shot verification of a composed system.

telecommunication is reported in [6].

2 Preliminaries

In this section we present some basic notions adopted from the work of Halbwachs et.al. which are used in the rest of the paper. Central to the modelling approach that we adopt is the concept of non-symmetric communication. This is different from the communication mechanism in e.g. process algebras [8] in that it allows an observer to observe the behaviour of a system without modifying it. Another model with a similar property is dynamic transition systems [9] which uses changes in the value of state variables for representing dynamic behaviour. In this paper we use the syntax of signal (event) based I/O machines – the prime motivation being the use of existing automatic verification tools, Mauto and Autograph for the synchronous language ESTEREL (which can be compiled to I/O machines).

2.1 Basic Definitions

Definition 1. Let S be a set of signals, and $E_s == 2^S$ be the set of events on S. An **I/O machine** M is a 5-tuple $(Q_M, q0_M, I_M, O_M, \delta_M)$ such that

- Q_M is the set of states containing $q0_M$, the initial state
- $I_M \subset S$, $O_M \subset S$ are the disjoint sets of input and output signals respectively
- $\delta_M \subseteq Q_M \times E_{I_M} \times E_{O_M} \times Q_M$ is the transition relation.

We require our specified systems to be *reactive*, i.e.

$$\forall q \in Q_M, \forall i \subseteq I_M, i \neq \emptyset \Rightarrow \exists (o, q') \text{ such that } (q, i, o, q') \in \delta_M$$

Such an I/O machine in response to a sequence $(i_1, i_2, \ldots, i_n, \ldots)$ of input events returns a sequence $(o_1, o_2, \ldots, o_n, \ldots)$ of output events, such that there exists a sequence $(q_0, q_1, \ldots, q_n, \ldots)$ of states with $q_0 = q0_M$ and $(q_{n-1}, i_n, o_n, q_n) \in \delta_M$ for all $n \geq 1$. The sequence $((i_1 \cup o_1), (i_2 \cup o_2), \ldots, (i_n \cup o_n), \ldots)$ will then be called a *trace* of the machine.

A *deterministic* machine has at most one possible reaction to a given input event. For a reactive deterministic machine M we denote by δ_M^O the function giving, for a state q and an input i, the output event o such that $(q, i, o, q') \in \delta_M$.

Definition 2. Let M be an I/O machine, and $O' \subseteq O_M$. The **projected machine** $M \downarrow O'$ is $(Q_M, q0_M, I_M, O', \delta')$, where $\delta' = \{(q, i, o \cap O', q') \mid (q, i, o, q') \in \delta_M\}$.

Definition 3. Let M_1 and M_2 be two I/O machines with $O_{M_1} \cap O_{M_2} = \emptyset$. The **synchronous product** of M_1 and M_2, denoted by $M_1 \parallel M_2$, is the I/O machine M where

- $Q_M = Q_{M_1} \times Q_{M_2},\ q0_M = (q0_{M_1}, q0_{M_2})$
- $I_M = (I_{M_1} \setminus O_{M_2}) \cup (I_{M_2} \setminus O_{M_1}),\ O_M = O_{M_1} \cup O_{M_2}$
- $((q_1, q_2), i, o, (q_1', q_2')) \in \delta_M \Leftrightarrow (q_1, (i \cup o) \cap I_{M_1}, o \cap O_{M_1}, q_1') \in \delta_{M_1}$ and $(q_2, (i \cup o) \cap I_{M_2}, o \cap O_{M_2}, q_2') \in \delta_{M_2}$

2.2 Verification of Safety Properties

In the verification approach based on I/O machines a safety property is considered as a set of traces. A *trace* σ on a set of signals S is a finite or infinite sequence of events on S. A property P on S is a *safety property* iff

$$\sigma \in P \Leftrightarrow \sigma' \in P \text{ for any finite prefix } \sigma' \text{of } \sigma$$

Central to the verification approach used here is the notion of *observer*.

Definition 4. Let P be a safety property on S. Let α be a (alarm) signal not in S. An **observer** of P is a machine $(Q_{\Omega_P}, q0_{\Omega_P}, S, \{\alpha\}, \delta_{\Omega_P})$ which is both deterministic and reactive with the following property. Let q_σ be the state that Ω_P reaches after reading the (possibly empty) trace σ. Then for any event $e \in 2^S$

$$\delta^O_{\Omega_P}(q_\sigma, e) = \begin{cases} \emptyset & \text{if } \sigma.e \in P \\ \{\alpha\} & \text{otherwise} \end{cases}$$

We further assume that all transitions with an $\{\alpha\}$ output event lead to a distinguished state q_α in Ω_P. Note that such a machine returns a sequence of empty output events as long as it receives a sequence of input events belonging to the safety property P. Moreover, the q_α state is not reachable from the initial state as long as the input traces are in P. Thus, the observer machine can be used for verification purposes as follows:

> An I/O machine M satisfies a safety property P iff the composed machine $M \parallel \Omega_P$ never outputs any event containing α.

2.3 Decompositional Verification

The above technique can of course be applied for verifying a program which in turn consists of a number of parallel modules. For verifying that $M_1 \parallel M_2$ satisfies the safety property P, one may construct the machine $M = M_1 \parallel M_2$ and ensure that $(M \parallel \Omega_P) \downarrow \alpha$ emits only empty output events.

Halbwachs et.al. [5], however, propose an additional method for decompositional verification which they suggest might be more efficient in some cases. The decompositional approach works as follows. Assume that we are interested in proving the safety property P in a system composed of modules M_1 and M_2. They suggest a two-step verification procedure which requires the additional notion of *restriction*.

For any $X \subseteq Q_M$, let $\widetilde{pre}_M(X) = \{q \mid \forall i \forall o \forall q' \text{ such that } (q, i, o, q') \in \delta_M,\ q' \in X\}$ be the set of states having *all* their successors in X.

Definition 5. Let M be an I/O machine and Ω_P an observer for a safety property P. Let $M' = M \parallel \Omega_P$. Then $sink_P$ is the set of states in M' leading inevitably to the violation of property P, and defined as follows:

$$sink_P = \mu X.\widetilde{pre}_{M'}((Q_M \times \{q_\alpha\}) \cup X)$$

where μ is the least fixed point operator.

Definition 6. Let M be an I/O machine and Ω_P an observer for a safety property P. Let $M' = M \parallel \Omega_P$ and assume $q0_{M'} \notin sink_P$. Then the **non-blocking restriction** of M', denoted by $M/_\infty\Omega_P$, is the I/O machine $(Q_{M'} \setminus sink_P), q0_{M'}, I_M, O_M, \delta'')$ where

$$\delta'' = \{(q, i, o, q') \in \delta_{M'} \mid q, q' \notin sink_P \text{ and } \alpha \notin o\}$$

Intuitively, the resulting machine has no traces with α events in, neither has it any states from which all the outgoing transitions (now pruned) had α events before the restriction. The two steps of decompositional verification can then described as follows. To prove that $M_1 \parallel M_2$ satisfies the property P on $S = I_{M_1} \cup O_{M_1} \cup I_{M_2} \cup O_{M_2}$, find a property P' on $I_{M_2} \cup O_{M_2}$ such that

1. M_2 satisfies P'
2. $M_1/_\infty\Omega_{P'}$ satisfies P.

That is, find a reasonable approximation to M_2 which has the necessary properties so that when composed with M_1 the required safety property is met by the combined system. An obvious application of this method is in the case of embedded systems. Then let M_1 correspond to our embedded software, M_2 correspond to its surrounding environment, and P' correspond to a relevant property of the environment. The proof then tells us that provided the environment of the software behaves as it should then the closed system has the required property.

This decompositional technique will be worthwhile only if each of these two steps is 'simpler' than the one shot approach of verifying the composed system. Our conjecture is that for a large class of embedded systems this is unlikely to be the case. The question is complicated by the fact that there is a potentially infinite number of possible intermediate properties P' that could be chosen. A crucial question is then: how to chose a suitable P'.

For example, assume we have a property P' which is thought to be suitable. If the second step above fails, then the cause of the failure may be in either of the two components (the module M_1 or the abstraction of M_2, $\Omega_{P'}$). If the property P' has been derived ad hoc then either M_1 or P' and ultimately M_2 may have to be changed. This can lead to an iterative process, where at each step the source of the current failure is tracked down and a fix attempted.

One way of avoiding these iterations would be if we could synthesize the weakest property P' which satisfied step 2 above. If such a property can be generated then we know that if step 1 fails then it is our model of the system, M_1 or M_2 that is at fault. In section 4 we look at a synthesis technique, and some of the technical details associated with it. First we present our example, and show how these verification techniques can be applied.

3 Example: Aircraft Landing Gear

In this section we use a model of a real world system to illustrate and assess observer-based verification. The system we have chosen to model is that taken from studies of the landing gear system of the Swedish JAS 39 Gripen multirole combat aircraft. This system has been modelled using a number of different formalisms, at various levels of complexity [10]. The earlier work included models where the relation between the physical properties of the hydraulic subsystem and the landing gear actuators could be studied. This enabled the study of the timeliness properties which are not covered here.

The model of the plant used in this paper is an abstraction of the physical models sufficient for the expression of a safety property. We consider the plant as being composed of two simple mechanical systems, a door and a gear, interacting with a software controller. The pilot can give a command for the gear to be extended or retracted. Extending the gear for example involves the door being opened, the gear being lowered, and then the door being closed again. The safety property which can be expresses in terms of synchronous observers is the property that the gear and door should never collide under operation.

3.1 Modelling

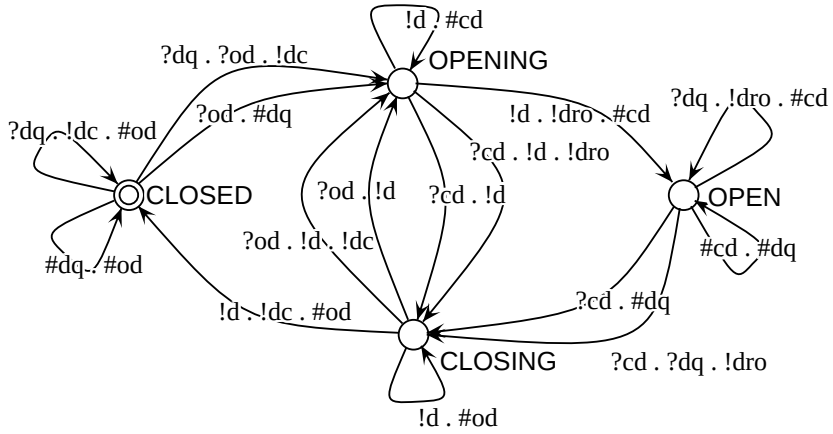
In this section we present various components of the system and discuss the composed model. To begin with a hybrid model of the plant was developed using the hybrid formalism of Delta IO machines [12]. The formalism represents a simple extension of discrete synchronous systems which allows the modelling of continuous processes within the framework of a standard synchronous language, in this case ESTEREL. In this paper we present the discrete structure of that model which contains sufficient information for verifying the safety property we are interested in.

The models are presented as I/O machines. They were generated directly from the ESTEREL compiler, and drawn using the Autograph package. Transitions on the graphs are signal operator pairs concatenated by full stops. The '!' operator represents that a signal is being output, a '?' operator represents signal input, and a '#' is an operator representing the absence of a signal from the input of a reaction.

The Plant For our purposes here, the 'plant' is considered to consist of everything beneath the software control system. This includes not just the physical system, but also the sensors, and the interface of the sensors with the controller.

Figure 1 shows an I/O machine representing the discrete fraction of the door model. This is half of the total plant model, the other half being the gear itself, which has very similar functionality (but will not be depicted here). The gear is raised and lowered in the same way that the door is opened and closed. The plant model is given by the parallel composition of the door and the gear models.

The machine in figure 1 has four states, the double ringed one representing the initial state. The two states shown here on the left and right of the diagram



inputs		outputs	
od	open door	dro	door open
cd	close door	dc	door closed
dq	door query	d	door in motion

Fig. 1. A model of the door as an I/O machine showing input and output signals

represent the fact that the door is stationary, and open or closed respectively. The top and bottom states represent the movement of the door.

The door can receive three inputs: **od** and **cd** are requests from the controller for the door to be opened and closed respectively, **dq** is a request from the controller for the door to report its state. As we can see from the transitions if the door is stationary (in states OPEN or CLOSED) and it receives a **dq**, it will report its current state with the outputs **dro** or **dc**, (door open and closed). If the door is in motion, the **dq** signal is ignored. On the transition from OPENING to the the OPEN state the door emits a **dro** without prompting, (and of course a **dc** for the corresponding CLOSING case). The signal **d** is a signal indicating the movement of the door. It is emitted spontaneously on an empty input whenever the position of the door is updated i.e. when the door moves.

So the operation of the door is essentially a cycle. It starts closed. A **od** will start it opening, once open a **cd** will start it closing again. At any stage its movement can be reversed by the appropriate command.

The Controller The controller is not depicted here because the state machine produced is too complicated to make easy sense of, consisting of 8 states and 48 transitions. The ESTEREL code for it is given instead in the Appendix.

The controller communicates with the plant via the ten control and sensor signals. In addition it receives two signals from the pilot, **cmd_up** and **cmd_down** representing commands to extend and retract the gear. It is assumed that the

controller should allow the pilot to disrupt the process of extension or retraction of the gear at any stage. It was also intended that the controller should not simply assume knowledge of the position of gear and door, but must check, every time a command is issued.

The System A diagram of the composed system is shown in figure 2. This model has been automatically generated using the parallel composition of the plant and controller models. From the initial state at the top, a **cmd_down** signal will start the door moving (opening) this can be seen on the graph as the transition **!d . #cmd_up**. Once the door is open the gear starts moving (lowering) and finally the door moves again (closing) and comes to rest. This is the progression down the left hand side of the diagram. At any stage if we receive a **cmd_up** signal, this will move control to the right hand side which brings the gear up.

We can clearly see just from this diagram that our composed system (8 states, 28 transitions) is simpler in terms of reachable states and transitions than either the plant model (16 states, 256 transitions) or the controller (8 states, 48 transitions) from which it is composed.

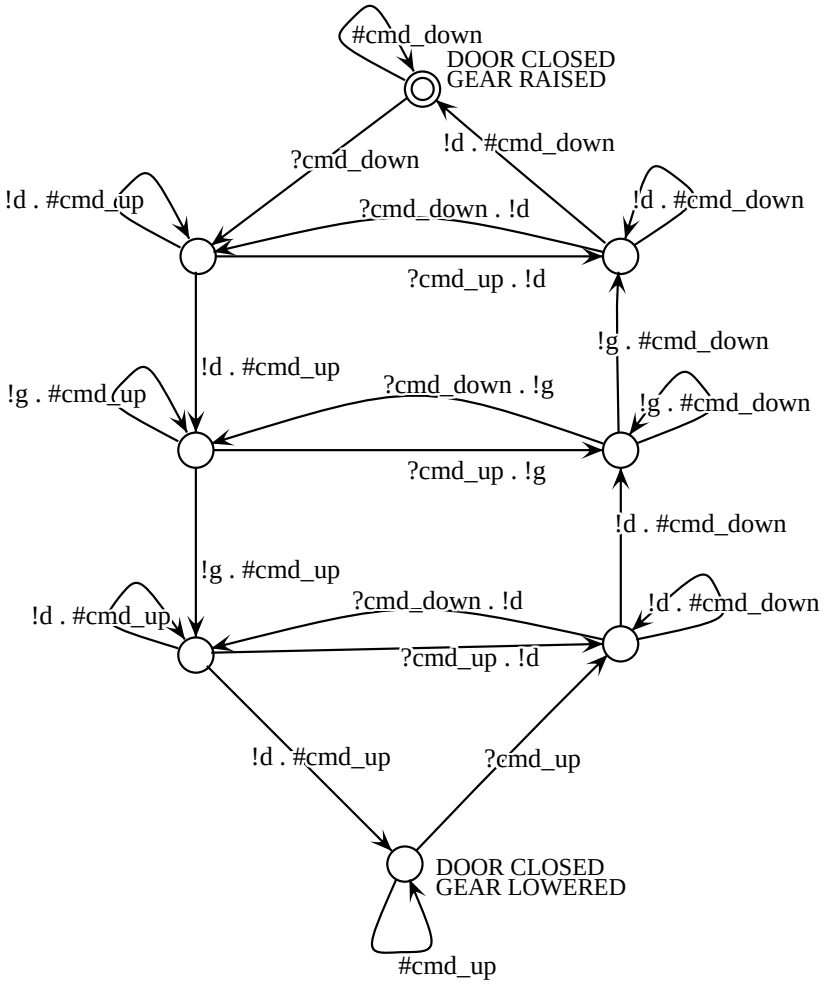
3.2 Compositional Verification

The property that we are interested in verifying is that the landing gear and the door will not collide. This property we prove by proving the stronger property that the door should always be open and stationary when the landing gear is in motion. This second property can be seen as two properties, that the door and gear are never moving at the same time, and that the door is always open when the gear is in motion. Notice that both properties, are properties of the composed system, and cannot hold over just the plant or controller.

These two properties can be checked by a pair of observer machines combined in parallel[4]. Figure 3 shows these machines. The first simply emits an alarm and halts whenever an event occurs containing the signals **g** and **d** at the same time, indicating that the door and gear have moved at the same time. The second machine watches the signals **dro** and **dc** which are sensor states from the plant emitted when the door is open and closed respectively. This second machine emits an alarm if **g** ever occurs when the sensors have indicated that the door is not open.

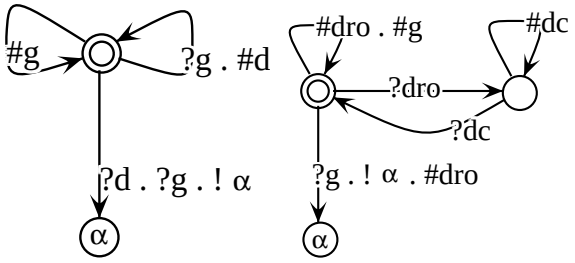
In order to check that the system satisfies the safety property, the complete observer is combined in parallel with the system, and the resulting state machine checked for the emission of α signals. Of course the proof is successful, and our composed system behaves as we require.

[4] These two machines share a common output α which technically means that they can't be combined using the product operator that was defined. This can be practically solved by calling the respective observers' outputs α_1 and α_2 and renaming both outputs to α in the composed machine.



inputs		**outputs**	
cmd_up	pilot command — raise gear	d	door in motion
cmd_down	pilot command — lower gear	g	gear in motion

Fig. 2. A model of the complete system. Communications between plant and controller could be shown here as outputs, but for clarity have been omitted.



inputs		outputs	
d	door in motion	α	alarm
g	gear in motion		
dro	door open		
dc	door closed		

Fig. 3. An observer of the safety property that the door and gear do not collide. The actual observer is the parallel composition of these two machines.

The complete observer has 4 states, 13 transitions and when combined with the system, the resulting machine has only 9 states and 35 transitions. This appeared to be a very small proof and the decompositional techniques would be hard pressed to improve on it.

3.3 Ad Hoc Decompositional Verification

In this subsection we consider verifying the property using an ad hoc approach to finding our intermediate property P'. The problem of synthesising this property automatically is complex, and we consider it in section 4. Here we consider finding a suitable property by using our intuitive understanding of the operation of the system.

There are two ways in which we can go about searching for P'. Either we can look for a property P' that is a property of the plant, or we can look for a property P' that is a property of the controller. In fact we tried both here.

The property is, of course, expressed as an observer. Finding suitable observers was a far from trivial task. We started by building observers that expressed what we thought might be sufficient conditions, and kept adding restrictions until we could make both steps of the proof hold. The resulting observers expressed much stronger properties than were necessary in the end. In the search for P' we were helped by the fact that we already knew that our composed system satisfied P. The task would have been much harder if we were trying to actually verify P for the first time, in which case proof failures could have been caused by errors in the system models as well as deficiencies in P'.

Both our observers were larger than the observer of the system property. The controller observer had 4 states and 61 transitions, its two verification steps

involved machines of size 19 states, 151 transitions and 13 states 212 transitions. The plant observer had 26 states and 2676 transitions and had verification steps with machines of sizes 27 states, 402 transitions and 61 states, 5108 transitions. Clearly both of these decompositions produced two much larger proofs than the verification of the composed system.

3.4 Does Decomposition Pay Off?

The example considered here is small, but large enough to be illustrative. From the work presented here we can conclude, that for similar cases:

- The composition of plant and controller can be smaller than either component.
- It is difficult to find an intermediate property suitable for decompositional proof by hand.
- Even when an intermediate property is found the two steps of the decompositional proof can involve larger constructions than the single proof of the composed system.

We tentatively conclude that this was not a suitable situation to apply decompositional verification.

Part of the problem that we experienced in applying the technique of decomposition to this example was due to the difficulty of finding a suitable intermediate property P'. The properties that we came up with in both cases were stronger than required. Ideally we would use the weakest possible P'. This would have the advantage that the two steps of the decompositional proof would provide necessary *and sufficient* conditions for our overall property to be satisfied by the composed system.

4 Weakest Observers

4.1 Synthesis Method

Halbwachs et.al. propose a method for synthesis of the weakest property P', and present a proof for the the following statement: if the property P' is arrived at in the proposed manner then it suffices to perform step 1 of the decompositional technique for proving the property P. To put this statement more precisely we restate a few auxiliary definitions.

Let $traces(Q)$ be the set of all finite traces $\sigma = e_1, \ldots, e_n$ such that $(q_{n-1}, i_n, o_n, q_n) \in \delta_M,\ i_n \cup o_n = e_n$, and $q_n \in Q$. For a trace $\sigma = (e_1, \ldots, e_n, \ldots)$ we define the *projection* of σ on a set of signals S' as the trace $\sigma \downarrow S' = (e_1 \cap S', \ldots, e_n \cap S', \ldots)$. The projection of a set of traces T on a set S' is defined as $T \downarrow S' = \{\sigma \downarrow S' \mid \sigma \in T\}$. Finally, if T is a set of finite traces on S, then $\mathcal{C}(T)$ is defined to be the set of traces on S which do not have any prefix in T.

To prove that $M_1 \parallel M_2$ satisfies P (denoted $\models P$), Halbwachs et.al. suggest the following method.

Proposition 7. *[5] Let Ω_P be an observer for P emitting α on violation of P. Let $T_{err} = traces(Q_{M_1} \times \{q_\alpha\})$ and let $P' = \mathcal{C}(T_{err} \downarrow S_2)$. Then*

$$M_2 \models P' \Leftrightarrow M_1 \parallel M_2 \models P$$

This proposition provides a method for doing decompositional verification that avoids the need for iterating, because the bottom line of claim 7 is an 'if and only if'. In other words if we generate the property in this way, then if P' is satisfied by M_2 the proof is successful, if it fails to be satisfied by M_2 then we know that one of the machines M_1 or M_2 is flawed.

Although this proof is expressed in terms of traces, it can clearly be implemented in terms of observers. The property $\mathcal{C}(T_{err} \downarrow S_2)$ can be expressed as an observer constructed from the synchronous product of M_1 and Ω_P, by changing signals from outputs to inputs (except α) and restricting the signals to those in S_2. This observer can then be used to check the properties of M_2 in the usual way.

4.2 Discussion

In comparing the different proof approaches we are interested in the size, that is number of states and transitions, of the I/O machines involved in the verification process. A decompositional approach might be considered worthwhile if it involves significantly smaller machines.

We have already observed that the composition of a plant and controller can result in a small machine. To see why we must consider the definition of synchronous product. For two machines A and B the size of their product $C = A \parallel B$ is related to the way in which the input to the composed machine is defined. Here its inputs are defined as $I_C = (I_A \setminus O_B) \cup (I_B \setminus O_A)$. Constraining the input alphabet in this way means that some potential transitions in C cannot exist because all their inputs have been removed from I_C. This situation will occur when the inputs of one machine are the outputs of another. Clearly this is the case in the situation of a plant and controller. In the extreme $I_A = O_B$ and $I_B = O_A$ in which case only spontaneous transitions i.e. those with empty input will survive.

In the case of our synthesized observer $\Omega_{P'}$, it is clear that it will be the same size as $M_1 \parallel \Omega_P$. Consequently the verification step on this machine which involves checking to see if $\Omega_{P'} \parallel M_2$ emits α, will involve comparable sized machines to the compositional verification step which checks $M_1 \parallel M_2 \parallel \Omega_P$ for α emissions. Not only is the last step comparable, the intermediate step it takes can also involve the unnecessary introduction of a large machine. Such an $\Omega_{P'}$ synthesized as a property of the plant in our landing gear example has 29 states and 547 transitions.

The observer $\Omega_{P'}$, as we have described it, is not of course the minimal weakest observer. We need to preserve any transitions in any path leading to an α-state. However any other transitions could be removed. This minimised version of $\Omega_{P'}$ could possibly provide smaller verifications. In practice, we found

that there were no opportunities for such minimisations when the techniques were applied to our example, since every transition was in a path leading to an α-state.

5 Summary

In this paper we have considered the use of synchronous observers for verification of embedded systems. We have considered a case where we have a property P that can not be ascribed to the plant or the controller alone – rather, it is a property of the composed system. To begin with, we note that starting to model the composed system is not to recommend for two reasons. First, to ensure correspondence with physical models, separate plant modelling is necessary. Second, checking the property of the composed system model does not give any hints as to what the controller should look like. Separate controller modelling leads to easier realisation of the control program code.

Next, we have considered alternative verification strategies. In this case it seems clear that a decompositional verification of the plant and controller can lead to harder proofs than a direct verification of the composed system, due largely to the tight interaction between the plant and the controller. It may be the case that alternative decompositions of the system into components which share fewer signals would be more fruitful. However such decompositions may not be a natural feature of the constructed hybrid models.

Finally a note on complexity of models. A significant problem being constantly attacked by the verification community is the state explosion problem. While efforts in this direction are essential for applications in the real world, we hope to have demonstrated in this paper that clever modelling can complement these efforts by keeping the size of the system under analysis to a minimum.

Appendix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Landing Gear Controller
%
%  Author: Martin Westhead
%
%  This is an implementation of a controller for a simple
%  landing gear model.
%
%  Inputs: cmd_up - retract gear, cmd_down extend gear,
%             dro - door open, dc  - door closed,gl - gear lowered,
%             gr - gear raised.
%  Outputs: od - open door, cd - close door, rg - raise gear,
%        lg - lower gear, gq qery gear state, dq - query door state
```

```
module CONTRL :

input cmd_up, cmd_down, dro,dc,gl,gr;
output od,cd,rg,lg,gq,dq;

await cmd_down; % nothing happens until pilot lowers gear
loop
  do
    emit gq;                % query state of gear
    present gl else         % gear is already lowered?
      run open_door;
      emit lg;              % send lower gear signal
      await gl;             % wait for gear to be lowered
      end;
    run close_door;
  up to cmd_up;

  do
    emit gq;
    present gr else         % gear is already raised?
      run open_door;
      emit rg;              % send raise gear signal
      await gr;             % wait for confirmation
    end;
    run close_door;
  upto cmd_down;
end
```

References

1. M. Abadi and L. Lamport. Composing Specifications. In J.W. de Bakker, W.P. de Roever, and G.Rozenberg, editors, *proceedings of the REX workshop on stepwise refinement of distributed systems, LNCS 430*, pages 1–41. Springer Verlag, 1989.
2. A. Børjesson, K.G. Larsson, and A. Skou. Generality in Design and Compositional Verification Using TAV. *Formal Methods in System Design*, 6:239–258, 1995.
3. H. De-Leon and O. Grumberg. Modular Abstractions for Verifying Real-time Distributed Systems. *Formal Methods in System Design*, 2:7–43, 1993.
4. R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Proc. Workshop on Theory of Hybrid Systems, October 1992, LNCS 736*, Lyngby, Denmark, 1993. Springer Verlag.
5. N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Third Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93*, Twente, June 1993. Workshops in Computing, Springer Verlag.
6. L.J. Jagadeesan, C. Puchol, and J.E. Von Olnhausen. A Formal Approach to Reactive Systems Software: A Telecommunications Application in Esterel. *Formal Methods in System Design*, 8:123–151, 1996.

7. R. Kaivola. Compositional Model Checking for Linear-Time Temporal Logic. In *Proc. 4th int. workshop on Computer Aided Verification*, pages 248–259. Springer Verlag, 1992.
8. R. Milner. *Communications and Concurrency*. Prentice-Hall, 1989.
9. S. Nadjm-Tehrani and J-E. Strömberg. From physical modelling to compositional models of hybrid systems. In *Proc. of the 3rd International Conference on Formal Techniques in Real-time and Fault-tolerant Systems*, pages 583–604. LLNCS 863, Springer Verlag, 1995.
10. S. Nadjm-Tehrani and J-E. Strömberg. Proving dynamic properties in an aerospace application. In *16th International Symposium on Real-time Systems*, pages 2–10. IEEE Computer Society Press, December 1995.
11. P.J.G. Ramadge and W.M. Wonham. The Control of Discrete Event Systems. *Proceedings of the IEEE*, 77:81–97, 1989.
12. Martin D. Westhead and John Hallam. Modelling hybrid systems as the limit of discrete computational processes. In *International Conference on Robotics and Automation*. IEEE, 1996.

This article was processed using the LaTeX macro package with LLNCS style