Compositional Verification of an Object-Based Model for Reactive Systems

M. Sirjani*, A. Movaghar*, M.R. Mousavi*

Department of Computer Engineering
Sharif University of Technology
Tehran,Iran

1 Introduction

Reactive systems have an ongoing interaction with their environment, accepting requests and producing responses. Correct and reliable construction of reactive systems is particularly important and challenging. So, using a formal method to establish mathematical proof of correctness of such systems is necessary.

A formal method consists of three major components: a model for describing the behavior of the system, a specification language to embody correctness requirements, and an analysis method to verify the behavior against the correctness requirements [8]. We choose an object-based model as the modelling language to describe the system, linear temporal logic as the specification language, and compositional verification for verifying that the model meets its specification.

In system verification, a main obstacle to the use of automatic methods is the state-explosion problem, which is the exponential increase in the number of system states caused by a linear increase in the number of system components or variables [6]. Compositional verification attempts to overcome the state-explosion problem by exploiting the modular structure that is naturally present in many system designs.

Compositional verification can decrease the complexity of problem when the model is naturally decomposable [10]. So, a model consisting of inherently independent modules is suitable for compositional verification. Object-based modelling is based on abstraction, encapsulation, and information hiding, using modules with high independence. Therefore, we choose an object-based model.

*emails: msirjani@mehr.sharif.ac.ir, { movaghar,mousavi}@sharif.ac.ir

2 Alecs Model

The model proposed here is similar to the actor model [2, 3, 4] in its independent active objects, asynchronous message passing, dynamically changing topology, and unlimited buffer for messages. This model does not support dynamic creation of active objects to decrease the complexity of the model and hence make the verification process easier. We call our model Alecs[1] and each active object alec. In Alecs one has to first define the classes and then introduce alecs as instances of that class. Often, we have several similar agents in a system that act concurrently and interact with one another. This feature allows us to reuse the class code in the model. Verification of a module specification and then composing the specifications is the basis of compositional verification; thus, in a model like ours, verification of a class property can be generalized to all of its instances. Therefore, we have reusability in the verification approach as well.

Independence of the alecs is important in compositional verification as well as in modelling. Alecs directly supports encapsulation and information hiding by separating the interface and the body of each alec's class definition. The interface part includes all the information necessary for interacting with other alecs and also for verification. Therefore, it is not just a modelling concern but a verification concern as well. This partitioning improves the readability too. Unlike other tasks in the verification of finite-state systems, which have been largely automated, current modular verification techniques still rely on user guidance [5]. As a result, readability is an important feature.

To overcome a common criticism on the actor model [4], an alec's semantics is defined such that the composition of two alecs is another alec.

The computation takes place by message passing and the execution of message servers. For each alec, there is an unlimited FIFO buffer called mailbox for arriving messages . There is a server (i.e. method) for each arrived message. When a message reaches the head of the mailbox its server is invoked, and the message is omitted from the mailbox and passed to the server. Each alec can send messages to its known alecs which are in the known alecs list. This list can be updated by sending alec's names in the messages. The topology of the model is based on this known alecs list.

The definition of the system includes the classes and the alecs in the running system. There is not a complete inheritance hierarchy, the classes act like templates for definition of the alecs. Alecs are instantiated from the classes and are composed in parallel to make the whole reactive system. The init server is invoked to initialize the system. Alecs are executed in parallel; each, having a single thread of execution, taking a message from the mailbox and invoking the server. If there is no message waiting in the mailbox then the alec will wait idle. Message passing is asynchronous and the sender alec continues its execution after sending the message. A part of computation and nondeterminism is modelled by asynchrony in message passing.

[1] Alive objects

3 Compositional Verification of Alecs

A necessary condition for compositional verification is that an alec's variable values shouldn't change when it is composed with other alecs, i.e. concurrent execution of alecs shouldn't change their specifications [1].

Executing message servers is the only way to change alecs' variable values. So, first we verify an alec property considering all possible environment conditions. It makes a state space presenting all the state transitions of the alec that can take place in the running system. Then it is obvious that safety property of an alec holds even if it is composed with another alec. So, because of alecs independence and encapsulation there is no need for explicit conditions in compositional verification of them.

We can suggest a strategy for compositional verification of safety properties. In this strategy, if P and Q are two alecs, and $\varphi_{P||Q}$ is a safety property of $P||Q$, we can show that $\varphi_{P||Q}$ is a property of compound alec, if we find safety properties for P and Q, such that

1. φ_P is a safety property of P,
2. φ_Q is a safety property of Q, and
3. $(\varphi_P \wedge \varphi_Q) \Rightarrow \varphi_{P||Q}$.

Also, we can use compositional minimization by finding the interface alec Q' corresponding to P. Then, we may model check a property on parallel composition of P and Q' $(P||Q')$, and conclude that the property is valid for $P||Q$ [9].

4 An Example

To show the power of Alecs model, we used it to specify a railroad control system, which has been used as a typical verification example [5, 7]. The system is made up of a bridge controller class(*BridgeController*) and a train class(*Train*). The bridge controller class prevents collisions between the two trains by ensuring the train safety requirement and the train class is a template for defining trains moving toward the bridge in different directions. Running system of this model consists of two train alecs(Train1 and Train2) and one controller alec(theController) which run concurrently as depicted in figure 1.

The system safety requirement, is that at any given time at most only one train is on the bridge. It is specified in temporal logic as follows:

$$\varphi_{sys} = [](\neg(Train1.OnTheBridge \wedge Train2.OnTheBridge))$$

We proved the correctness of above statement using minimization rule on safety properties of $BridgeController$ and $(BridgeController||Train)$. The state space of the system, with traditional verification methods, consists of 288 states. However, applying compositional verification results in just 56 states.

```
ActiveClass BridgeController {
  Interface :
     KnownObjects :
        T[1..2] : Train;
     MsgServers :
        Arrive(sender),
        Leave(sender);
     Observational :
        Signal[1..2]: (green, red) ;

  Body :
      isWaiting[1..2]: Boolean;
      Arrive(sender)
      {
       theOtherOne: Integer;
       theOtherOne := (sender  mod 2) + 1;
        if Signal[theOtherOne] := red  then
        {
          Signal[sender]:= green;
          send (sender, YouMayPass);
         }
        else
            isWaiting[sender] := true;
        }

      Leave(sender)
      {
        theOtherOne: Integer;
        theOtherOne := (sender mod 2) + 1;
        Signal[sender] := red;
        if  isWaiting[theOtherOne] then
         {
               send (T[theOtherOne],
                                YouMayPass);
         }
        isWaiting[sender] := false;
       }
      Init()
      {
       Signal[1] := red; Signal[2] := red;
       isWaiting[1] := false;
       isWaiting[2] := false;
      }
    }
```

```
ActiveClass Train{
  Interface :
    KnownObjects :
      Controller: BrigdeController;
    MsgServers :
      YouMayPass(),
      ReachBridge(),Passed() ;
    Observational :
      OnTheBridge : Boolean;

  Body :
      ReachBridge()
      {
       send(Controller, Arrive(self));
      }

      YouMayPass()
      {
       OnTheBridge:= true;
        send(self,Passed);
      }

      Passed()
      {
       OnTheBridge:=false;
       send(Controller,Leave(self));
       send(self,ReachBridge);
      }

      Init()
      {
       send(self,Passed);
       OnTheBridge:= false;
      }
     }
     ActiveObjects:
       Train1, Train2: Train;
       TheController: BridgeController;
     Composition:
       Train1 || TheController || Train2
     Init()
     {
      Train1.Controller := theController;
      Train2.Controller := theController;
      theController.T[1] := Train1;
      theController.T[2] := Train2;
      }
```

Figure 1: A railroad control system in Alecs

5 Concluding Remarks

In the future, we plan to add real-time constraints to the model, increase the reusability of the model, and add the possibility of dynamic creation of alecs. Since decomposing the system into modules, and specifying the properties are currently the designer's responsibility, we shall make them as automatic as possible. The first step is to present some guidelines and algorithms in this area. In this paper, we focus on safety properties only, verification of progress properties may also be considered. Induction methods and special optimization algorithms for model checking of the Alecs will also be needed in the future.

References

[1] Abadi L. and Lamport L., Composing Specifications, in *ACM Transactions on Programming Languages and Systems*, vol. 15, No. 1, pp. 73-133, January 1993.

[2] Agha G. and Hewitt C., Concurrent Programming Using Actors, in Yonezawa A., Tokoro M. Eds.,*Object-Oriented Programming*, MIT Press, pp. 37-53, 1988.

[3] Agha G., The Structure and Semantics of Actor Languages, *Foundations of Object-Oriented Languages: REX School Workshop*, LNCS 489, pp. 1-59, 1991.

[4] Agha G., Mason I., Smith S. and Talcott C., A Foundation for Actor Computation, *Journal of Functional Programming*, No. 7, pp. 1-72, 1997.

[5] Alur R. and Henzinger T.A., *Computer Aided Verification*, Draft, 1999.

[6] Alur R., de Alfaro L., Henzinger T.A. and Mang F.Y.C., Automating Modular Verification, in *Proceedings of the Tenth International Conference on Concurrency Theory*, LNCS 1664, Springer-Verlag, pp. 83-97, 1999.

[7] Bjorner N.S., Manna Z., Sipma H.B. and Uribe T.E., Deductive Verification of Real-time Systems Using Step, in *Proceedings of ARTS-97*, LNCS 1231, Springer-Verlag, pp. 22-43, 1997.

[8] Manna Z. and Pnueli A., *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, 1992.

[9] Peng H. and Tahar S., A Survey on Compositional Verification, *Technical Report Dept of ECE*, Concordia University, 1998.

[10] Roever W.P., The need for Compositional Proof Systems: A Survey, in Roever W.P., Langmaack H. and Pnueli A., Eds., *Compositionality: The Significant Difference*, LNCS 1536, Springer-Verlag, pp. 1-22, 1998.