Compositional Reasoning for Port-based Distributed Systems

Alma L. Juarez Dominguez
School of Computer Science
University of Waterloo
Waterloo, ON, Canada
aljuarez@cs.uwaterloo.ca

Nancy A. Day
School of Computer Science
University of Waterloo
Waterloo, ON, Canada
nday@cs.uwaterloo.ca

ABSTRACT

Many distributed systems using IP-based communication protocols consist of chains of components that run concurrently and communicate asynchronously with their neighbours through ports. We present a compositional reasoning method using model checking and theorem proving to verify liveness properties of a communication protocol for chains of connections consisting of an unknown number of components. We outline how our method is used to verify properties of the call protocol of AT&T's Distributed Feature Composition (DFC) architecture.

Categories and Subject Descriptors: D.2.4 [Software/ Program Verification]: Formal methods

General Terms: Verification.

Keywords: Compositional verification, DFC, model checking, theorem proving.

1. INTRODUCTION

Many distributed systems consist of independent components that run concurrently and communicate asynchronously via first-in, first-out queues. The protocols used often create graphs of connections where a component communicates only with its immediate neighbours. For example, in AT&T's Distributed Feature Composition (DFC) architecture [10] for IP-based telecommunication services the components are telecommunication features such as call waiting and call forwarding. In this work, we are interested in proving liveness properties of a protocol for chains of connections of an unknown number of finite state processes that communicate asynchronously. Our contribution is in exploiting domain-specific information about the properties to be proven and the system to produce a compositional reasoning method consisting of a combination of theorem proving, model checking, and language containment.

First, we recognize that while the components of the system are heterogeneous, to satisfy the protocol properties, they must all satisfy the same properties. We use the term *semiregular* to describe a set of components that behave the same with respect to the protocol but have distinct individual behaviours. This observation allows us to decompose the overall system properties into identical obligations that each individual component must satisfy. These individual properties are verified by model checking each component, and the properties are combined using induction in a theorem prover. The theorem proving effort is only in terms of properties of the components and properties of the queues, and does not need to be repeated as more components are added to the system.

The individual properties are unlikely to hold in every environment, but rather only when the component is placed in an environment of a chain of similar components using the same protocol. Here, we exploit a second aspect of domain-specific knowledge. The components in a chain communicate only with their immediate neighbours. A *port* is the communication behaviour that a component has with one neighbour sending and receiving messages. If we prove that a component works in an environment consisting only of the ports of its neighbours, then it will work in the entire chain. The port serves as an abstraction of not only its immediate neighbour but all components on that side of the chain. Furthermore, because of the semiregularity attribute, we are able to find an abstract representation of a port to serve as the environment in which we check individual components rather than checking all combinations of a component with possible neighbouring ports. We use language containment to show that the behaviour of any port is contained within the behaviour of the abstract model of a port.

By exploiting these two domain-specific attributes, we have created a compositional reasoning method to verify liveness properties of a protocol used by chains of connections of an unknown number of components in a semiregular, port-based, asynchronous distributed system with bounded queues. We demonstrate the utility of our approach by describing a significant case study showing how our method works to prove protocol properties of the DFC architecture.

2. COMPOSITIONAL REASONING

The goal of our work is to verify overall system properties of a communication protocol in all chains of connections with any number of components. We decompose the problem into two main parts: verify properties of individual components, which we will call *individual properties*, and then assuming all components satisfy the individual properties, prove the

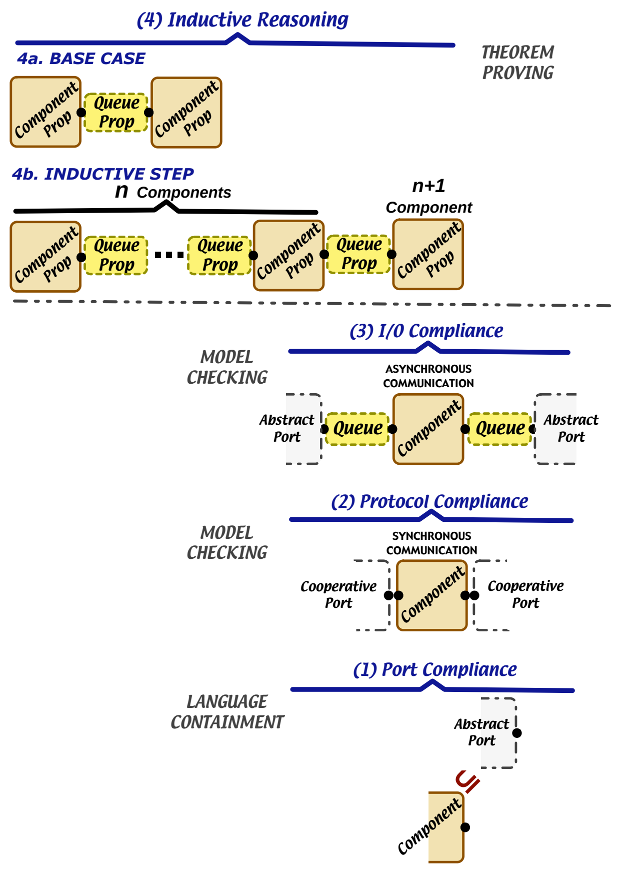

Figure 1: Compositional Reasoning Method

overall system property. Figure 1 illustrates our compositional reasoning method. Verifying components individually is done in steps (1), (2), and (3) using model checking and language containment. Proving the overall system property is done using induction over the structure of the chain of components in a theorem prover in step (4).

The proof obligations of steps (1), (2), and (3) allow us to conclude that a component will satisfy the individual properties in *any* chain of components of the system. These steps must be completed for every component. Because the system is semiregular and uses port-based communication, we create an abstract model of a port and can use this abstraction to represent the behaviour of the neighbouring port plus all aspects of the chain on the other side of the neighbour. The abstract model of the port captures that most general behaviour of a port. In step (1), which we call *port compliance*, we verify that the behaviour of each port of every component is included in the behaviour of this abstract port.

Unlike broadcast communication, port-based communication over chains of components means that a component will only communicate with its immediate neighbours through a fixed maximum number of ports. We decompose the step of verifying the protocol properties of a component into two parts to reduce the state space. First, we show *protocol compliance* in step (2), which means that the component satisfies the individual properties. Second, we show *I/O compliance* in step (3), which means that the output produced by the component is expected by its environment and the environment provides the input expected by the component. We expect the I/O compliance step to be reusable for multiple individual properties (and therefore multiple overall system properties). Next, we provide further details on these steps.

In the *port compliance* step, we prove, using language containment, that the behaviour of each port in a component is within the behaviour of the abstract port. The language of the ports is the communication between the port and the queue. First, we isolate the behaviour of the component to its communication on only one port by replacing all transition triggers except those dealing with communication on this port with a guard of "true" and removing all outputs except those for the port being verified. This is a valid abstraction of the port's behaviour – it does not add or remove any port behaviour. Second, we find an abstraction function, abs, matching the states of the component (`concrete`) with the states of the abstract port (`abstract`). Then we show, for every transition in the concrete machine consisting of a source state (src), destination state (dest), and a trigger (sig), which involves receiving or sending a signal, that:

$$\forall\ \mathsf{src},\ \mathsf{sig},\ \mathsf{dest} \cdot (\mathsf{src},\ \mathsf{sig},\ \mathsf{dest}) \in \mathtt{concrete}$$
$$\Rightarrow (\mathsf{abs}(\mathsf{src}),\ \mathsf{sig},\ \mathsf{abs}(\mathsf{dest})) \in \mathtt{abstract}$$

In the *protocol compliance* step, we use model checking to verify the individual properties of a component using cooperative ports and synchronous communication. A *cooperative port* sends and receives any signal the component needs during its execution. It allows us to hide the behaviour of the environment by assuming the port will cooperate. The use of synchronous communication abstracts away the behaviour of the queues, which considerably reduces the state space during the verification effort.

In the *I/O compliance* step, we use model checking to verify that a component communicating with neighbouring components asynchronously over a channel of bounded length receives only the signals it is expecting and sends only the signals expected by the environment. In this step, we use abstract ports as the neighbours of the component. We check for a combination of lack of deadlock (invalid end states) and termination with empty queues. No other components in the chain need to be considered because the component only communicates with its immediate neighbours.

The size of queues needed between the components for checking I/O compliance will depend on the system being verified. In our DFC case study, we required a queue size of only one to prove lack of deadlock and termination with empty queues. While a larger queue size could have been used, the smaller queue size reduces the state space and forces the maximum number of interleavings. Since we check that the component and its environment are deadlock free, this channel size is sufficient.

In the *inductive reasoning* step, we use induction in a theorem prover to prove the overall system properties. The induction is on the structure of the chain of components. We assume the individual properties hold of all components and that the queues provide perfect communication: any message that is sent on a queue will eventually reach the component's neighbour. The base case is two components connected by a queue. In the inductive step, we assume the overall system property holds if there are n components in the chain, and then prove the overall system property will hold if there are $n + 1$ components in the chain. The inductive reasoning is performed in terms of properties only, and is only performed once. Since we are not working with models of components in this step, there is no state space search.

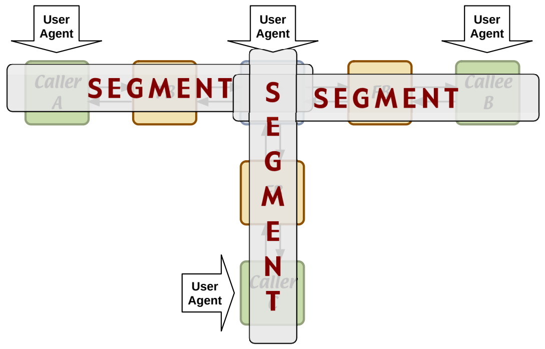

Figure 2: Usage composed of Segments

Currently, we use the SPIN model checker [9] because it supports both synchronous and asynchronous communication and, therefore, we describe the properties in linear temporal logic (LTL). We use the HOL theorem prover [6], and a tool that we wrote for checking language containment.

3. CASE STUDY: DFC

In this section, we describe how our method is applied to the DFC architecture, developed by Jackson and Zave at AT&T for coordinating telecommunication features [10]. A *feature box* is a function for the users of a system that is performed on top of basic services. A DFC *usage* is a graph that describes the response to a request for a telecommunication service. The nodes of the graph are features and the edges are bidirectional communication channels, which are first-in, first-out queues. The port connected to the box that initiates a call is called a *caller port*, and the port on the other end of the channel is called a *callee port*.

A *box* is a process that performs interface or feature functions. *Interface boxes* provide an interface to physical devices. *Feature boxes* are either free (an instance of it is generated to be included in a usage) or bound (an instance of it is dedicated to a particular address, so the same feature box is made part of any usage). We have modelled interface boxes, three free feature boxes, and the bound feature box call waiting (CW) as processes in PROMELA, the modelling language of SPIN. Our complete model and proofs are available in [12]. The most complicated process is CW, and its state transition diagram contains 338 states and 445 transitions.

Some DFC boxes can change the topology of a usage by placing, receiving, or tearing down calls. To describe properties of DFC chains, we characterized boxes as:

***User agents* (*UA*):** A box that can request the creation of a chain of components, or respond to such a request. A user agent can create a branch in a chain.

***Transparent* (*T*):** A box that must forward any call protocol signals that it receives.

We denote as a *segment* any part of a usage that is a chain of components that starts and ends at a user agent box. Segments can be connected together at user agents, assembling a branching usage, as illustrated in Figure 2.

The *call protocol properties* of segments provide a concise statement of end-to-end behaviour of DFC, where the "ends" are user agents. The segment properties are of two forms: **(1)** Signals that propagate from one *UA* to another *UA* in one direction; and **(2)** Signals that propagate to the end of a segment only if another signal of the same kind has not been sent by the other end and received at some intermediate box. By checking all the segments that compose a usage, we verify the behaviour of the usage.

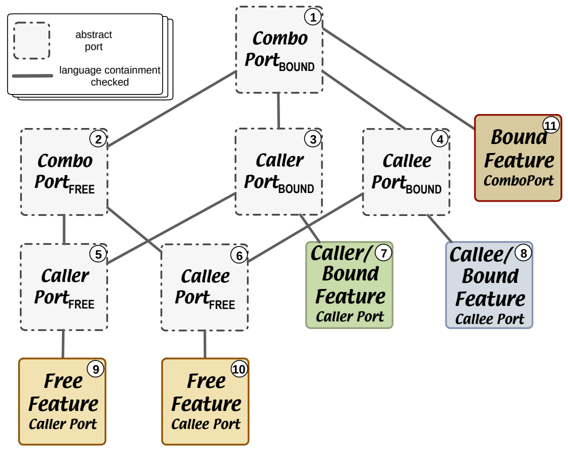
abstract port
language containment checked
1 Combo Port$_{BOUND}$
2 Combo Port$_{FREE}$
3 Caller Port$_{BOUND}$
4 Callee Port$_{BOUND}$
11 Bound Feature ComboPort
5 Caller Port$_{FREE}$
6 Callee Port$_{FREE}$
7 Caller/ Bound Feature Caller Port
8 Callee/ Bound Feature Callee Port
9 Free Feature Caller Port
10 Free Feature Callee Port

Figure 3: Abstract Models of Port Behaviour

Now we describe how our compositional method is used to verify the segment properties on DFC segments of unknown, but finite length. Here we just provide the DFC specific details, the rest is as explained in Section 2.

Protocol Compliance. First, we determined the individual properties that capture the essential box behaviour by hand, although this was quite straightforward since they reflect what an individual box must do so that the segment properties will be satisfied. We check these properties in SPIN on the caller process, callee process, and four feature boxes that we have modelled. The verification of CW took the longest amount of time at 20 seconds.

Port Compliance. The abstract model captures the essential behaviour of a DFC port. We started using the caller and callee port models presented in the DFC manual as our abstract ports [11], but found that CW ports can switch between being a caller and a callee during the box's execution. We arrived at an abstract model of a port, called a *combo* port, manually through trial and error, but it is the union of the behaviours of every component's ports.

The abstract models can be arranged in a partial order based on language containment as shown in Figure 3, where the dashed boxes are the abstract models (1-6), and the shaded boxes (7-11) represent the ports of particular boxes. We checked the behaviour of the boxes we modelled against the appropriate abstract port model, so we know that as long a box's port behaviour is contained within one of the abstract models, it is contained within the most general abstract model. This part of the verification took 5 seconds for each CW port.

I/O Compliance. We use SPIN to verify the I/O compliance property: that each feature box placed in an environment of the most abstract ports (*i.e.,* combo bound ports) only receives signals it expects and sends only the signals the abstract port models expect. We checked for a combination of lack of deadlock (invalid end states) and termination with empty queues automatically using XSPIN. It took 5 seconds to check the I/O compliance property for the CW box.

Inductive Reasoning. Since we have completed the protocol compliance, I/O compliance and port compliance steps, we know the individual properties hold of each DFC box within a chain of components. We assume that the queues behave perfectly. Then, we restrict our reasoning to only consider the DFC individual box properties in LTL to prove the segment properties for all segments of an unknown, but finite length n. Because we reason about segments, the base case is two UA boxes communicating through a queue, whereas the inductive step is a proof for n transparent boxes delimited by UA boxes to conclude a segment with $n + 1$ transparent boxes delimited by UA boxes (all communicating through queues).

4. RELATED WORK

Most existing compositional reasoning techniques (*e.g.,* [4, 14]), including assume-guarantee reasoning [15, 7, 8, 1] have focused on verifying systems with synchronous communication between a fixed number of heterogeneous components. Induction in a theorem prover can be used to reason about an unknown, but bounded number of identical components. Theorem proving has also been used to justify the reduction of the system to a fixed number of identical components that can be model checked (*e.g.,* [13, 16]). We are interested in systems consisting of an unknown number of components with asynchronous communication, taking advantage of the attribute of semiregularity of the components with respect to the protocol.

Handling asynchronous communication offers challenges not usually addressed by assume-guarantee style reasoning, which focuses on synchronous or broadcast communication. Many systems have effectively unbounded queues because they send messages on the internet. Proving properties of systems with unbounded queues is undecidable in general [3]. In the future, we hope to build on work that characterizes decidable subsets of this problem (*e.g.,* [5]). Our use of theorem proving may offer us some advantages in being able to describe the required properties of queues abstractly.

There have been other efforts to verify DFC-related artifacts [17, 2]. These efforts checked for lack of deadlock. We check call protocol properties and provide a compositional approach for checking the properties on segments of unknown length. Similar to our work, the verification effort of [2] consists of combining a feature with standardized environmental peer entities of caller, callee and dual ports, however, they only check for lack of deadlock using synchronous communication. We extend this work by checking liveness properties, as well as taking the step of showing that all box behaviour is contained within an abstract model, which captures the most general DFC port behaviour. We also present a partial order among these abstract models.

5. CONCLUSION

We have described a compositional reasoning method for protocol properties of port-based distributed systems with chains consisting of an unknown number of components and communicating using bounded queues. We demonstrated how the method can be used to verify liveness properties of the DFC call signalling protocol. Our verification method allows us to reason about components individually, which considerably reduces the verification effort. We use inductive reasoning to show that the overall system properties hold for chains with an unknown number of components. This form of compositional reasoning is possible when a system with port-based communication has the attribute of semiregularity, which is common for protocol properties.

6. REFERENCES

[1] N. Amla, E. A. Emerson, and K. S. Namjoshi. Efficient decompositional model-checking for regular timing diagrams. In *Correct Hardware Design and Verification Methods*, volume 1703 of *LNCS*, pages 67–81, 1999.

[2] G. Bond, F. Ivancić, N. Klarlund, and R. Trefler. ECLIPSE feature logic analysis. *IP-Telephony Workshop*, pages 49–56, 2001.

[3] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[4] J. Burch, E. Clarke, and D. Long. Symbolic model checking with partitioned transition relations. In *International Conference on Very Large Scale Integration (VLSI)*, pages 49–58. IFIP Trans., North-Holland, 1991.

[5] X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and verification of reactive electronic services. In *Int'l Conf. on Impl. and Application of Automata*, number 2759 in LNCS, pages 188–200. Springer, 2003.

[6] M. J. C. Gordon and T. F. Melham. *Introduction to HOL*. Cambridge University Press, 1993.

[7] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Trans. on Prog. Lang. and Sys.*, 16(3):843–871, 1994.

[8] T. Henzinger, S. Qadeer, and S. Rajamani. You assume, we guarantee: Methodology and case studies. In *Computer-Aided Verification*, volume 1427 of *LNCS*, pages 440–451. Springer-Verlag, 1998.

[9] G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

[10] M. Jackson and P. Zave. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Trans. on Soft. Eng.*, 24(10):831–847, Aug. 1998.

[11] M. Jackson and P. Zave. *The DFC Manual*. AT&T Labs, Nov. 2003.

[12] A. L. Juarez Dominguez. Verification of DFC call protocol correctness criteria. Master's thesis, School of Computer Science, University of Waterloo. May, 2005.

[13] K.Bhargavan, D. Obradovic, and C. A. Gunter. Formal verification of standards for distance vector routing protocols. *Jour. of the ACM*, 49(4):538–576, Jul. 2002.

[14] K. L. McMillan. Verification of infinite state systems by compositional model checking. In *Correct Hardware Design and Verification Methods*, number 1703 in LNCS, pages 219–233. Springer, 1999.

[15] A. Pnueli. In transition from global to modular temporal reasoning about programs. In *Logic and models of concurrent systems*, pages 123–144. Springer, 1985.

[16] A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Learning to verify safety properties. In *International Conference on Formal Engineering Methods*, volume 3308 of *LNCS*, pages 274–289. Springer, 2004.

[17] P. Zave. Formal description of telecommunication services in Promela and Z. In *19th International NATO Summer School: Calculational System Design*, pages 395–420, 1999.