Applied GIS

ISSN: 1832-5505

Vol-9 Issue-04 Dec 2021

Symbolic Model Validation for Component Connectors Based on Channels

P.Beersheba [1], Ch.Ambedkar [2],

Assistant Professor [1],[2],

Department of CSE, SRK INSTITUTE OF TECHNOLOGY ENIKEPADU VIJAYAWADA

Mail Id : shebapolimetla@gmail.com, Mail id : rahul59985@gmail.com,

Abstract

This study details the theoretical underpinnings and empirical findings of a model checker for channel-network-modeled component connections in the Reo calculus. To reason about the data flow and coordination rules in a network, the specification formalisms use a branching time logic. The Model checking for CTL-like logics is based on versions of traditional automata-based methodologies. In this implementation, binary decision diagrams are used to symbolically depict the network and the available I/O-operations. The effectiveness of our model checker has been shown by applying it to a few cases

Keywords: binary decision procedures, data streams, constraint automata, and branching time logicDiagrams

introduction

Over the last 15 years, several coordination languages and models have emerged, each of which offers a formal description of the glue code used to connect components and which may be used as a springboard for formal verification. For the exogenous coordination language Reo [2], we focus on the latter component here. Through a chain of operations that generate channel instances and connect them in (network) nodes, Reo's glue code is acquired as a network of channels.

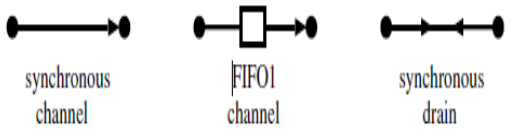
Reo network semantics have been supplied in a variety of coherent forms.

Accept and offer predicates codify whether and which data items may be written or read at a node, respectively, across a variety of network setups, as described in [2]. The timed data stream semantics of [5] is demonstrated to be compatible with the operational semantics of a Reo network provided in [6] using a version of labelled transition systems called constraint automata. While Reo is a beautiful formalism for synthesizing component connections using simple composition operators, it may be challenging to make sense of Reo networks that have numerous nodes and channels. Therefore, a vital part of using the Reo framework for complicated situations is having tool support for assessing the coordination mechanism described by a Reo network. The (bi)simulation and language equivalence testing algorithms in [6] and the temporal logic specification algorithms in [3,10] are examples of verification algorithms for Reo networks based on their constraint automata semantics.

Reo Constraint and automata

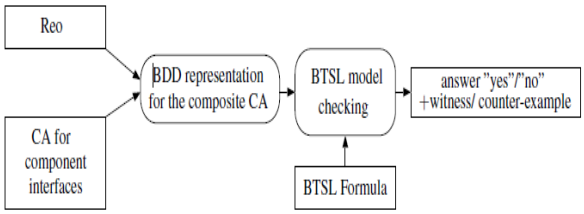
Here, we provide a high-level overview of Reo, a coordination language with a semantics based on operational constraint automata.References [2,6] provide further information. In Reo, an exogenous coordination language, complicated component connections are formed in a compositional way by organizing in a network of channels. When it comes to coordinating and interacting with other nodes in a network, the glue code is provided by Reo networks. Reo uses a loose concept of channels and allows for any form of peer-to-peer interaction. The channels in a Reo network need to have a user-defined semantics and two channel endpoints that are either sink or source ends. Data is written into the channel at the source end and read out at the sink end.

ISSN: 1832-5505
Vol-9 Issue-04 Dec 2021



Three basic kinds of channels that will be used as examples are graphically shown in the image above. There is a source and a drain in synchronous and FIFO channels, respectively. Writing and reading must be done at the same time through synchronous channels.done at the same time. The FIFO channel seen in the center has a single buffer cell and is hence referred to as a FIFO1 channel. As long as there is no data in the buffer, writing from the source end is permitted. When you write d, it is saved in the buffer. If the buffer is full, data may be read from the sink end, and the item will be removed from the buffer. The synchronous drain provides a powerful route for the development of sophisticated Reo coordination principles. It has two inputs but no outputs. Both ends of a data item are being erased at once, making simultaneous writing impossible.

Sets of channel terminations are represented by the nodes in a Reo network. There are three types of nodes that result from Reo's join operator: source nodes, sink nodes, and mixed nodes. Each type is determined by whether or not all of the channel ends that coincide on a node A are source ends, sink ends, or a combination of both. Connecting components to a network via input and output ports, or source and sink nodes. The admixed vertices Logic that Forks in Real Time Here, we provide a temporal logic based on branching time, which may be used to reason about the control and data flow in a constraint automaton. This reasoning, known as Branching Time, CTL [11,12], PDL [15], and TDSL [3,9,4] are all components of Stream Logic (BTSL). Similar to CTL, formulae may employ the route quantifiers and to refer to the configurations of a component connector (states of a constraint automaton) using atomic propositions ap AP. Until, a standard operator for expressing path attributes, or the PDL/TSDL-like modality __, where is a regular expression providing sequences of I/O-operations at the nodes, are the two most common ways to define a path. BTSL, you'll need a tuple (AP,N) where AP is a collection of atomic propositions and N is a set of nodes. BTSL syntax is divided into three tiers, marked by the capital Greek letters and for state formulas, the tiny greek letter for run formulas, and the letter for normal I/O-stream expressions. Model Validation in Symbolic BTSL 4 As input, the BTSL model validation issue requires a Reo network, which may also includes a BTSL formula that has to be verified, and constraint automata that describe the interfaces of the components that make up the links between the network's source and sink nodes. Connected system components' automata to the network's sink or source nodes, the environment in which the network functions is described. Since certain transition instances (concurrent I/O-operations) may become impossible owing to the behavioral interfaces of the components, they may limit the non determinism in the automata for the network. When node A, which is both a sink and a source, is connected to a port on a component, A is said to be a mixed node. As a result, the component's automata might also reduce the number of possible terminal states. When these automata are ignored, the analysis will take into consideration all possible interactions between the sink and source nodes even if nothing is known about the prospective behaviors of the components that will be managed by the network.



network, maybe within of the ecosystem provided by the automata for the components.

The second stage is to prove or disprove that a certain BTSL formula is true for all beginning states of the created constraint automata. Yes, without a doubt forms of formulas The model checker may either provide a witness (such as a run with |=) or a counterexample (such as a run with _|=) for the formula being tested. We provide a symbolic BTSL model checker in the next section. After briefly outlining the fundamentals of the BTSL model checking technique, we move on to describe the symbolic implementation of this method. Five Instances, With Outcomes A handful of instances were run via the BTSL model checker. Here, we will report on There are two examples of this. All testing was performed on a Pentium IV with 1.8GHz of processing power and 1.5GB of RAM running Mandriva Linux with kernel 2.6.12. The program was developed in C++ and GCC4.0.3 was used to build it;

ISSN: 1832-5505
Vol-9 Issue-04 Dec 2021

the JINC [18] library was used to create the binary decision diagrams. 5.1 (Philosophers at Dinner) as an Example The first provides a Reo model of the classic philosophers' dinner (see Fig. 5 in [1]).

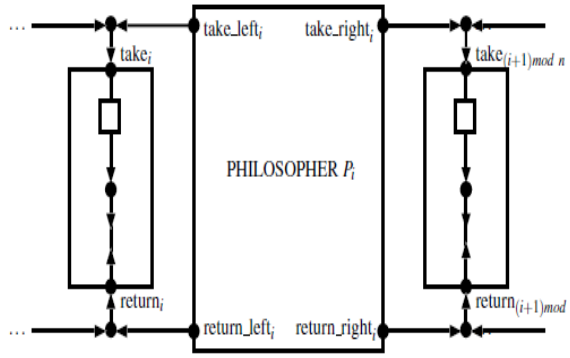


Fig. 5. Dining philosophers scenario

...to the philosopher's right. An FIFO1 channel and synchronous drain represent the chopsticks in this analogy. Figure 6 depicts the philosopher and chopstick interface constraint automata.

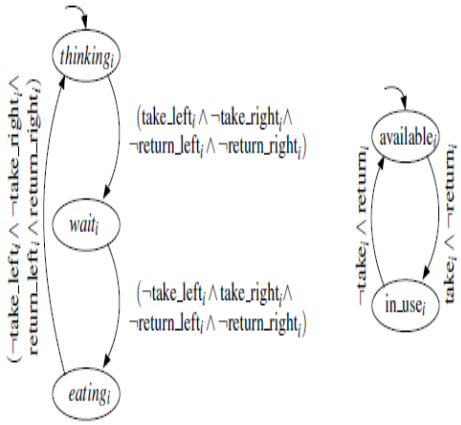


Fig. 6. CA for philosopher and chopstick

In Table 1, we see how effectively the symbolic join-operation can be used to build the BDDrepresentation of the constraint automaton A for the whole system. The "size" column provides a count of philosophers. The In the table below, "time" represents the amount of time required during synthesis, whereas "reachable time" indicates the amount of time required to calculate the reachable fragment of A. The other two columns detail the largest BDD created throughout the symbolic computation and the size of the BDD generated for A.Since there is a run in which all philosophers accept the left chopstick and then wait indefinitely for the missing right chopstick, the second formula cannot hold. Using a backward iteration of 798 times, we have discovered this impasse. analysis. The stalemate may be identified in only 403 steps and 13.92 seconds using forward analysis to compute the reachable section first. Mutual exclusion: Example 5.2 The component connection seen in Fig. 7t is used as an example two.

Size	Time	BDD Nodes	Peak	Reach Time
200	0.98s	33146	285523	0.24s
400	2.18s	66546	572523	0.45s
800	4.97s	133346	1146523	0.86s
1600	12.69s	266946	2294523	1.81s
3200	35.12s	534146	4590523	3.96s
6400	112.21s	1068546	9182523	8.53s

Table 1
Synthesis results for the dining philosophers

hat provides a critical action bottleneck of no more than k processes per time instance for a set of n parallel processes (P1,..., Pn)

ISSN: 1832-5505
Vol-9 Issue-04 Dec 2021

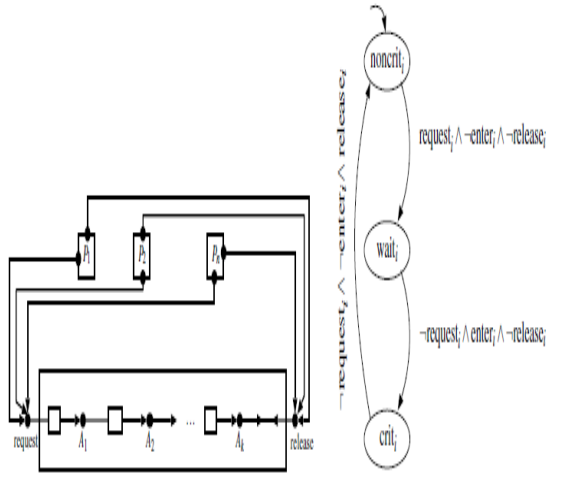


Fig. 7. Mutual exclusion scenario and CA for one process

.

In this case, we assume that each component's Pi has a behavioral interface represented by the constraint automaton shown in Fig. 7. The generated BDD-representation findings are summarized in Table 3. where n is the total number of processes and k is the maximum permissible critical section occupancy. There are more than 5 10119 possible configurations in this CA if you have 200 processes and k = 60.

The Final Six

The study aimed to provide light on the reasoning behind and operation of our Reo network model checker. Two examples have been provided to demonstrate the speed with which our model verification method can process extremely large networks with up to 101200 configurations. We feel that our model checker offers a significant contribution for formal reasoning regarding exoge neous coordination models, especially in light of the vast variety of applications of the Reo framework (see, for example, [13,20,9]). Our implementation will be expanded to reason about real-time constraints using the logic TDSL [3] or a branching time version thereof, and about dynamic reconfigurations using the logic considered in [10] or other formal frameworks for Reo's dynamic operations, in addition to further efficiency optimizations and case studies.

References

[1] F. Arbab, Abstract Behavior Types: A Foundation Model for Components and Their Composition, In [7],33-70, 2003.

[2] F. Arbab, Reo: A Channel-based Coordination Model for Component Composition, Mathematical Structures in Computer Science, 14(3):1-38, 2004.

[3] F. Arbab and C. Baier and F. de Boer and J. Rutten, Models and Temporal Logics for Timed Component Connectors, In Proc. SEFM'04, IEEE CS Press, 2004.

[4] F. Arbab and C. Baier and F. de Boer and J. Rutten, Models and Temporal Logics for Timed Component Connectors, Software and Systems Modelling (to appear), 2006.

[5] F. Arbab and J.J.M.M. Rutten, A coinductive calculus of component connectors, In Proc. 16th WADI

volume 2755 of LNCS, pages 35-56, 2003.
[6] C. Baier and M. Sirjani and F. Arbab and J.J.M.M. Rutten, Modeling Component Connectors in Reo by Constraint Automata, Science of Computer Programming, 61:75-113, 2006.

[7] F.S. de Boer and M.M. Bonsangue and S. Graf and W.-P. de Roever, Formal Methods for Components and Objects, LNCS 2852, Springer, 2003.

[8] R. Bryant, Graph-Based Algorithms for Boolean Function Manipulation, IEEE Transactions on Computers, C-35, 1986.

[9] D. Clarke and D. Costa and F. Arbab, Modeling Coordination in Biological Systems, In Proc. of the Int. Symposium on Leveraging Applications of Formal Methods, 2004.

[10] Dave Clarke, Reasoning about Connector Reconfiguration II: Basic reconfiguration Logic, In Proc. FSEN'05, Teheran, Electronic Notes in Theoretical Computer Science, 2005.

[11] E. Clarke and E. Emerson and A. Sistla, Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, ACM Transactions on Programming Languages and Systems, 8(2):244-263, April 1986.

[12] E. Clarke and O. Grumberg and D. Peled, Model Checking, MIT Press, 1999.

ISSN: 1832-5505
Vol-9 Issue-04 Dec 2021

[13] N. Diakov and F. Arbab, Compositional Construction ofWeb Services Using Reo, In Proc. International Workshop on Web Services: Modeling, Architecture and Infrastructure (ICEIS 2004), Porto, Portugal, April 13-14, 2004.

[14] E. Emerson and C. Lei, Modalities for Model Checking: Branching Time Strikes Back (extended abstract), In Proc. 12th Annual ACM Symposium on Principles of Programming Languages (POPL), pages 84-96, SIGPLAN, ACM Press, 1985.

[15] M. Fischer and J. Ladner, Propositional dynamic logic of regular programs, Journal of Computer and Systems Sciences, 18:194-211, 1979.

[16] G. Hachtel and F. Somenzi, Logic Synthesis and Verification Algorithms, Kluwer Academic Publishers, 1996.

[17] K. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.

[18] J. Ossowski, JINC, a bdd library (to be published), www.jossowski.de

.

[19] I. Wegener, Branching Programs and Binary Decision Diagrams. Theory and Applications, Monographs on Discrete Mathematics and Applications, SIAM, 2000.