A Tool for Deadlock Analysis of Parameterized-chain Networks

M. Moodi, M. H. Zibaeenejad*, J. G. Thistle

Department of Electrical Engineering, University of Waterloo, ON, N2L 3G1, Canada (e-mail: mmoodi,mhzibaee,jthistle@uwaterloo.ca)

Abstract: This paper studies algorithmic aspects of deadlock analysis for parameterized networks of discrete-event systems. A parameterized network consists of interacting finite-state subsystems, including finite but arbitrarily large numbers of subsystems within each of a finite number of isomorphism classes. While deadlock analysis of such systems is generally undecidable, decidable subproblems have recently been identified. The decision procedure of Zibaeenejad and Thistle (2017) rests on the construction of a finite *dependency graph* for the network, and the computation of its *full, consistent* subgraphs. We present a software tool that takes the template of a Parameterized Chain Network (PCN) and outputs the set of all full, consistent subgraphs of the dependency graph. These subgraphs represent infinite set of deadlocked states of the PCN for all parameter values. As a case study, we investigate deadlock in a complex train network that extends beyond the current theoretical framework. The results suggest ways in which the framework could be extended.

Keywords: Discrete-event systems, Parameterized systems, Deadlock analysis, Model-checking

1. INTRODUCTION

A fundamental impediment to analysis and design of complex control systems is combinatorial explosion: the state size of a collection of N interacting subsystems grows exponentially with N. One potential means of coping with state explosion is to devise methods that are independent of N. Vector discrete-event systems Li and Wonham (1993) and Petri nets support such methods to some extent for certain classes of systems.

In this paper, we employ the model of *parameterized-chain networks* proposed in Zibaeenejad and Thistle (2017, 2015). A *parameterized system* is generally a family of finite-state models, indexed by some parameter or parameters. In a relatively simple case, a parameterized network might be a model of N isomorphic, interacting, finite-state subsystems, where the value of the parameter N is finite but arbitrarily large; this indeed yields an infinite set of finite-state network models, indexed by the values of the integer parameter N.

When a system reaches a state from which there are no possible transitions, it is said to be in a total deadlock. If a subsystem reaches a state from which no further transitions are possible, regardless of any events that may occur within the larger system, the overall system is said to be in a partial deadlock. One method of detecting deadlocks is an exhaustive search over the global system state set (Chandy et al. (1983)). But this method is not suitable for a parameterized network with an infinite state-set.

Model checking *cut-offs* sometimes apply in deadlock analysis of parameterized networks whose subprocesses fall into a finite number of isomorphism classes. Relatively small bounds, 'cut-offs', can, under some conditions, be established, such that if the property of interest is verified successfully in all instances limited by the cut-offs, the satisfaction of the property in a parameterized network is guaranteed. Unfortunately, results of this nature are based on restrictive models of subsystem interaction, such as the unidirectional passing of tokens that carry no data, around a ring network (Emerson and Kahlon (2000, 2004, 2002)). In contrast, the approach of (Zibaeenejad and Thistle (2017)) uses the notion of weak invariant simulation to impose directionality of control flow. This allows the modeling of systems such as traffic networks and manufacturing systems that cannot be modelled under the assumptions on which cut-offs are based. Specifically, (Zibaeenejad and Thistle (2017)) investigates existence of deadlocks in networks comprised of a fixed set of finite-state *distinguished* subprocesses, linked by parameterized networks of linear topology. Such networks can model, for example, transportation networks and manufacturing systems. The notion of *weak invariant simulation* is used to impose a direction of control flow on the network, and this structure, together with additional assumptions, ensures decidability of the existence of reachable "generalized circular waits." To locate these circular waits, a finite *dependency graph* is constructed, which shows how the possibility of execution of an event within a given subprocess of an instance of the network, may depend on the occurrence of events within another subprocess. The generalized notion of a "circular wait" is captured by a subgraph of the dependency graph that has the properties of *consistency* and *fullness*. Such a subgraph represents in general an infinite set of circular-wait states that can arise in network instances of various sizes; in effect, it represents a regular language, each element of which encodes a generalized circular wait in a particular size instance of the network. The present paper discusses algorithmic aspects of the approach. In particu-

lar, it discusses the construction of the dependency graph and the computation of consistent, full subgraphs of the dependency graph. For technical reasons related primarily to analysis of the reachability of generalized circular-wait states, the theoretical analysis of (Zibaeenejad and Thistle (2017)), is restricted to a special case in which the network has a unique, distinguished "input subprocess." This restriction is ignored in the present paper, and our software tool is used to perform experimental research that extends beyond the current theoretical framework and will support extension of the theoretical work.

2. PRELIMINARIES

2.1 Discrete Event Systems Basics

One of the conventional ways of presenting a DES employs *generators* (Wonham (2012)). A nondeterministic generator is formally defined as a 4-tuple $G = (X, \Sigma, \xi, x^0)$, where X is a state set, Σ a finite alphabet representing a finite event set, $\xi : X \times \Sigma \to 2^X$ is a transition function (where 2^X is the power set of X), and x^0 an initial state. A shared event between two generators is an event that is enabled from states of these generators. It can occur if both of the generators are in states that allow the shared event: transitions labeled by a shared event occur simultaneously in generators that share the event. Local events are not shared with any other generator. The semantics of shared and local events are formalized by means of *synchronous products* (Wonham (2012)).

2.2 Linear parameterized discrete event systems

A Parameterized Discrete Event System(PDES) $\mathcal{P}$ is an infinite set of synchronous products of M isomorphic finite-state subprocesses, where M ranges over the set of natural numbers greater than two. Formally,

$$\mathcal{P} = \{\|_{i=1}^{M} P_i : \ M > 2\},$$

where $P_i = (X_i, \Sigma_i, \xi_i, x_i^0)$, with $X_1 = X_2 = ...$, and M is the unspecified parameter. We are particularly interested in PDES with *linear* topology. PDES $\mathcal{P}$ has linear topology if for any member $\|_{i=1}^{M} P_i \in \mathcal{P}$, subprocess P_i, $1 < i < M$, has events shared only with both P_{i-1} and P_{i+1}, and P_1 and P_M respectively have events shared only with P_2 and P_{M-1}.

We assume all subprocesses have the same state set X_s and instantiated from a template subprocess P_n in the following manner. Let $P_n = (X_n, \Sigma_n, \xi_n, x_n^0)$, and assume all event symbols in Σ_n have either n or $n+1$ as indices. Define instance P_i for any $i \in \mathbb{N}$, by replacing the index n (respectively $n+1$) with i (respectively $i+1$), and defining ξ_i such that for all $x \in X_s$ and $\sigma_n \in \Sigma_n$ (respectively $\sigma_{n+1} \in \Sigma_n$), $\xi_i(x, \sigma_i) = \xi_n(x, \sigma_n)$ (respectively $\xi_i(x, \sigma_{i+1}) = \xi_n(x, \sigma_{n+1})$).

We set $\Sigma_i = \Sigma_{L_i} \cup \Sigma_{S_i}$; Σ_{L_i} is the set of local events (events that are shared neither with P_{i-1} nor with P_{i+1}) and Σ_{S_i} is the set of shared event symbols. Local event alphabets are pairwise disjoint. Symbols in Σ_{S_i} either have index i or index $i+1$: shared events between subprocesses P_{i-1} and P_i have index i, while event shared between P_i and P_{i+1} have index $i+1$.

2.3 Parameterized-chain networks

Formally, a *PCN* is a strongly connected, finite, directed graph whose nodes are partitioned into *distinguished* nodes and *parameterized* nodes (see figure 1 for an example). The former, represented graphically as squares, will denote distinguished subprocesses, and the latter, represented as circles, will denote linear PDES that are subnetworks of the overall system. Distinguished nodes are finite-state distinguished subprocesses (a distinguished subprocess can have a structure distinct from those of other subprocesses). Each parameterized node is the template finite-state subprocess for the linear PDES that the node denotes. All parameterized nodes have an in-degree and an out-degree of one. We assume that the state sets corresponding to subprocesses associated with different nodes are disjoint. We denote the (distinguished) nodes with in-degree larger than one *input nodes*, and the nodes with out-degree larger than one *output nodes*. For the formal definition PCN and assumptions on a PCN see (Zibaeenejad and Thistle (2017)).

2.4 Deadlock analysis scheme

The deadlock analysis rests upon the construction of a dependency graph of the PCN. This is a finite graph whose nodes are states of distinguished subprocesses or of the template subprocesses corresponding to linear parameterized subnetworks. The presence of an edge from one such node to another indicates that when neighbouring subprocesses within an instance of a PCN are occupying those those respective states, then the first subprocess cannot execute any event until the second subprocess has executed an event that is shared with another neighbouring subprocess - the first subprocess is dependent on the second in order to proceed. Consistent subgraphs represent sets of states of subsystems of network instances. As such, they include at most one state of any distinguished subprocess (hence the name 'consistent'). They are also required to include a state of at least one input subprocess in each of their maximal strongly connected components - this rules out subgraphs that only represent states of parameterized segments of the network. To capture circular waits, we also require that each node within a consistent subgraph belong to a strongly connected component of that subgraph - in other words, that a strongly connected subgraph be the union of its strongly connected components. But the notion of a circular wait must be generalized to account for the branching in our network topologies. For this, we define a full consistent subgraph to be one where, for each node that is a state of an output subprocess, and for each direct successor subprocess of that output subprocess, if there exists a shared event between the two subprocesses that can occur when the output subprocess is in its given state, then the subgraph must contain an edge from the state of the output subprocess to some state of the direct successor subprocess. The fullness property thus ensures that in any states represented by the consistent subgraph, each output subprocess is 'dependent' on every one of its direct successors with which it could conceivably execute a shared event. The reader is referred to (Zibaeenejad and Thistle (2017)) for more formal definitions and results.

3. THE SOFTWARE TOOL

Our program will draw the dependency graph and all of its maximal consistent and full subgraphs. To generate the PCN graph and the automaton model of each node in the PCN, we use Integrated Discrete-Event Systems Software (IDES v.3 beta 1 [1]). IDES generates XML files with the extension of .xmd, and these files are used as inputs to our program.

The program first reads and parses the input PCN XML file and all of the other automaton model XML files. Then it finds all of the parameterized and distinguished nodes in the PCN graph [2]. The software instantiates the parameterized nodes by creating two more nodes and XML files for each parameterized node. After that, it finds all of the simple loops in the PCN graph using a depth-first search algorithm. The respective automaton models of the nodes in the loop are then modified by deleting all transitions labeled by events shared with parts of the network that are not included in the loop. In each loop, the software calculates the synchronous product of each node with its next neighbor node in the PCN graph. Each state in the synchronous product is an ordered pair of states of the automata corresponding to the respective nodes of the PCN graph. The algorithm then finds the states of the synchronous product where the only enabled events are shared with the successor of the second component of the synchronous product (within the loop). All of the found states from the synchronous product will form the nodes and edges of the dependency graph. Each state is an ordered pair of nodes and in the dependency graph the first node in the pair will be connected to the second by an edge. After forming the dependency graph the tool locates all full, consistent subgraphs of the dependency graph.

To find all of the maximal full, consistent subgraphs of the dependency graph we will find all of the simple loops in the dependency graph; then we will assign a number to each of these simple loops and call them s 'super-node'. We will create an undirected graph, named the super-graph, with each node being one of these super=nodes. We assign an edge between any of these nodes if the corresponding simple loops have at least a common node in the dependency graph. Now using the super-graph, we can investigate consistency and fullness properties. To have a consistent subgraph, there are two conditions that must be satisfied by each subgraph:

(1) Any strongly connected component of the subgraph must contain at least one input node.
(2) There should not be more than one state of each distinguished node in any subgraph.

To satisfy the above conditions, we create the adjacency list of the nodes in the super-graph. We traverse this graph and eliminate those paths that have more than one state of a given distinguished node in their simple loops and that do not include any input node. At the end we will generate all of the maximal consistent subgraphs. To satisfy the fullness condition, we find all of the output nodes' states in each simple loop of the dependency graph and check if in the PCN graph those output nodes in that specific state have any shared events with their neighbors. If there is any shared event with a successor in the PCN graph, to satisfy fullness, there should be an edge from that specific output node to that successor in the corresponding subgraph of the dependency graph. So, to have maximal consistent and full subgraphs, we eliminate generated maximal consistent subgraphs that do not satisfy the fullness property.

[1] `https://qshare.queensu.ca/Users01/rudie/www/software.html`
[2] The user must specify the name of the parameterized nodes that have in-degree and out-degree of one.

4. CASE STUDY: COMPLEX TRAIN NETWORK

In this section we consider a more complex version of the train network example of (Zibaeenejad and Thistle (2017)). The model consists of distinguished subprocesses that represent the intersections within the network and of linear parameterized segments representing routes of arbitrary length. It would be seen that the existence of generalized circular waits will depend on the lengths of these routes.

Figure 1 is the PCDN graph and represents a network of six intersections and seven routes. A train will enter the network from the IA_1 "input node" and travel through the network using any of the specified routes. (Directions of the movements are indicated by the arrows) Each space in the network will get filled by the arrival of a train and will empty upon its leaving. IA_1, IB_1 and IC_1 are the input nodes and IA_2, IB_2 and IC_2 are the output nodes. Rx_i, $R'x_i$, $R''x_i$ (x could be A, B and C, except $R'B_i$), R_i and R'_i are the parameterized nodes. Distinguished nodes have been represented using square nodes in the graph. The traffic network represented in figure 1 could be arbitrarily large with many distinguished and parameterized nodes. For the purpose of this example the network contains seven routes, each represented by a parameterized node, which means their length could be arbitrarily large. Each intersection is also represented by a distinguished node. To be consistent with the simpler version of this network we will assume every train comprises two cars and will occupy two spaces at a time. In order to satisfy the assumptions underlying the reachability analysis of (Zibaeenejad and Thistle (2017)), trains are modeled as entering input nodes in a single event; they are also modeled as leaving the network in a single event. Otherwise, they pass through spaces on the routes one wagon at a time. Each intersection is blocked after the entrance of a train and will accept new trains only upon the departure of the first train. Suppose that a train enters the network from intersection IA_1, and then continues to the main route. Upon arrival at the next intersection, IA_2, it has three choices: it could go to the upper or lower route to come back to intersection IA_1, or it could continue to reach to the next intersections, IB_1 and IB_2. Again the train has three different choices. It could continue to the main route to return to the IA_1 intersection, or it could choose the lower or upper route. By choosing the lower route, it will come back directly to the IB_1 intersection, but by choosing the upper route it will enter the next intersection, IC_1, and again there is a choice to make, upper route or lower route, both of which will eventually reach the next intersections IC_2 and IB_1. We will instantiate the PCDN network using three subprocesses in each parameterized segment, which will allow us to analyse fully the parameterized network $Zibaeenejad and Thistle$ (2017).

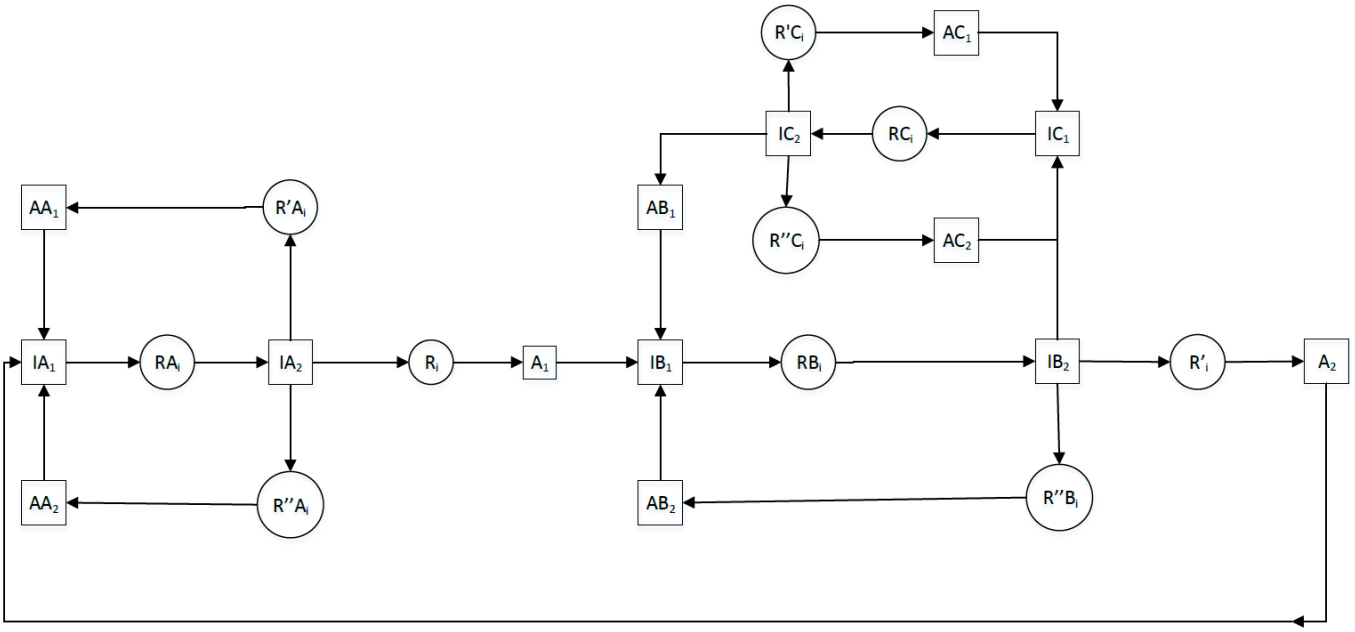


Fig. 1. PCN graph of train network consisting of 10 parameterized nodes (depicted as circles) and 14 distinguished nodes (depicted as squares). Parameterized nodes model routes with arbitrary lengths. Distinguished nodes model intersections or parts of routes that may require separate modeling.

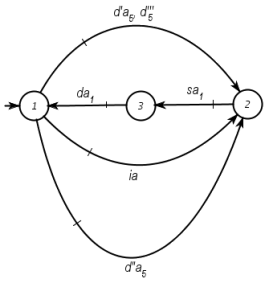


Fig. 2. The input node IA_1

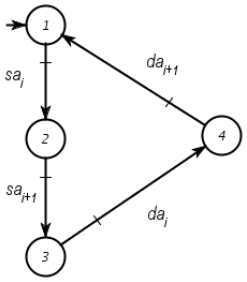


Fig. 3. Parameterized node representing the main route

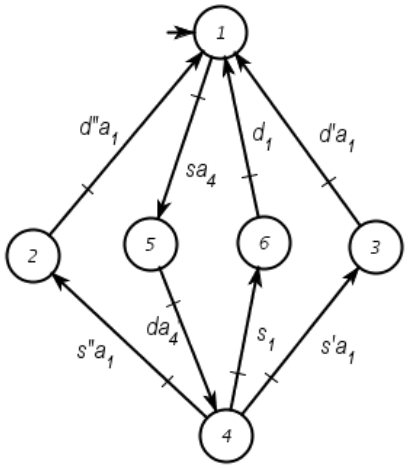
All of the input nodes including IA_1, IB_1 and IC_1 intersections, have a structure similar to figure 2, with a slight relabeling of event names (replacing 'a' in IA_1 to 'b' or 'c', to achieve IB_1 and IC_1 graphs respectively.) In intersection IA_1, the entrance of the train from outside of the network has been denoted by a local event named ia. The only difference among these input nodes is that IB_1 and IC_1 do not have the local events ib and ic, respectively because trains cannot enter the network from those intersections. Their shared events with the previous space are denoted by d'''_5, d_5 and d''_5 in IA_1, IB_1 and IC_1, respectively. Shared events $d'a_5$, $d''a_5$ and d'''_5 represent entrance of the train from top, bottom and previous space of the main route, respectively. This intersection will be emptied when the first and second wagons of the train leave this intersection by sa_1 and da_1 events, respectively. The main route after the first input node has been represented by a parameterized node RA_i.

In figure 3 the first and second wagon will arrive at the i^{th} space by events sa_i and da_i and they will leave by events sa_{i+1} and da_{i+1} respectively. Other routes which have been modelled by a parameterized node have the same structure to this one and can be constructed by proper relabeling of the above model.

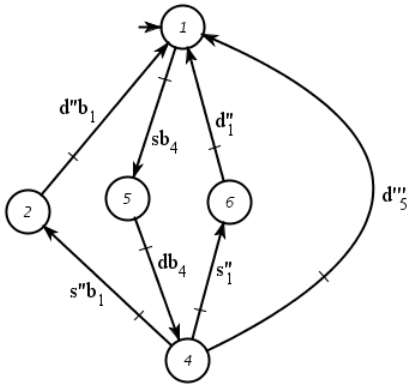
The intersection IC_2 has the same structure with a slight relabeling of the event names.

Figure 4a depicts the structure of the output nodes IA_2 and IC_2. The first and second wagon will enter this intersection via events sa_4 and da_4 from the main route. After that wagons can leave this intersection in three different ways; they could go to the lower route using $s''a_1$ and $d''a_1$ events or they could continue along the main route using s_1 and d_1 events or finally they could exit to the upper route using $s'a_1$ and $d'a_1$ events, which is the only difference between IA_2 and IC_2, on the oner hand, and IB_2 on the other hand, as depicted in figure 4b. Intersection IC_2 has the same structure as IA_2 because they are both connected to a parameterized node in their upper route, while IB_2 is connected to a distinguished node in its upper route, and input nodes can accept trains in just one event.

Figure 5 depicts the models of the other distinguished nodes in the PCDN network. All of the distinguished nodes depicted in figure 5a are the same except for a slight change

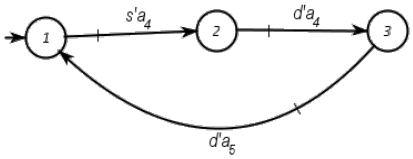


(a) IA_2 and IC_2

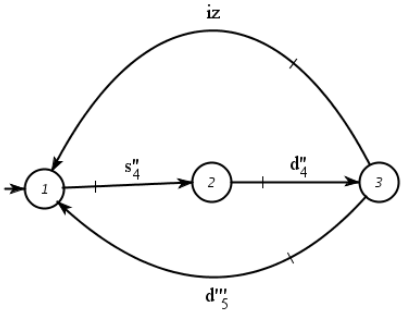


(b) IB_2

Fig. 4. Output nodes



(a) AA_1, AA_2, AB_1, AB_2, AC_1, AC_2, A_1



(b) $A2$

Fig. 5. Distinguished nodes

in event names. This space gets filled by entrance of two wagons via $s'a_4$ and $d'a_4$ events and become empty by the $d'a_5$ event. The only difference in the A_2 node is that it has one more local event for trains to leave the network, as depicted in 5b.

4.1 The dependency graph and its full, consistent subgraphs

Figure 6 represents the dependency graph of the traffic network generated by the developed software tool. This dependency graph contains eight distinct consistent and full subgraphs. To satisfy the consistency condition, subgraphs must include an input node and also they must contain just one state of each distinguished node. For example, in figure 6, the loop between nodes $2_{R'A}$ and $4_{R'A}$ does not satisfy the first condition of the consistency condition, so it does not represent a circular wait. The state set of the nodes are represented by a state number in the dependency graph. For example, in 3_{IA_1}, the state of the IA_1 has been represented by 3, while in $4R$, the state of the all of the parameterized nodes are 4. In figure 6, a consistent subgraph cannot contain both nodes 3_{IA_1} and 2_{IA_1} or nodes 3_{IC_1} and 2_{IC_1}, because a subprocess cannot be in two different states at the same time. As depicted in figure 7, all of the components of the subgraph satisfy the consistency conditions by including an input node and not having more than one state of a distinguished node. It also represents the satisfaction of the fullness condition by having an edge from the output node 2_{IC_2} to $2_{R''C}$. Because that output node has an event shared with its neighbour $R''C$ in the PCN graph. The first component of the subgraph in figure 7, requires even number of parameterized nodes for the 2_{RB} and 4_{RB} loop; while the loop containing the nodes $2_{R''B}$ and $4_{R''B}$ requires an odd number of parameterized nodes to reach a partial deadlock in the network. In other words, the partial deadlock can occur only if the number of spaces on the lower return route from I_{B_2} back toward I_{B_1} is odd. By the same reasoning we should instantiate the other parameterized nodes in the second component of the subgraph, 2_{RC}, 4_{RC}, $2_{R''C}$ and $4_{R''C}$ with an odd number of parameterized nodes; to have a deadlock in the network. That is the number of spaces on the lower route RC from I_{C_1} to I_{C_2} and on the lower return route from I_{C_12} to I_{C_1} must be odd. We altered some of the configurations of the automaton models, to see how the changes would affect the results. First, there is no difference in the results when the trains can enter the network via any of the input nodes rather than just via one input node (I_{A_1}). Secondly, if the we let the train to leave the network also via node AB_1, in the dependency graph and subgraphs there would be no state of the AB_1, as expected. Each such subgraph represents a partial deadlock and if the whole network does not have any of the states represented by these subgraphs, then it is free of any deadlocks.

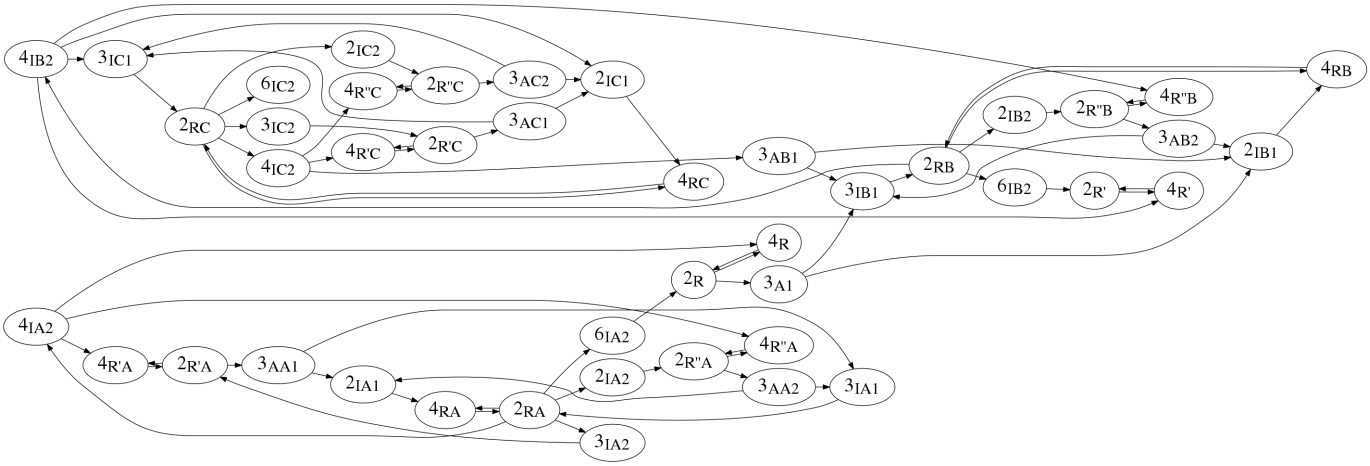


Fig. 6. Dependency graph, generated by the developed software tool for deadlock analysis of parameterized network.

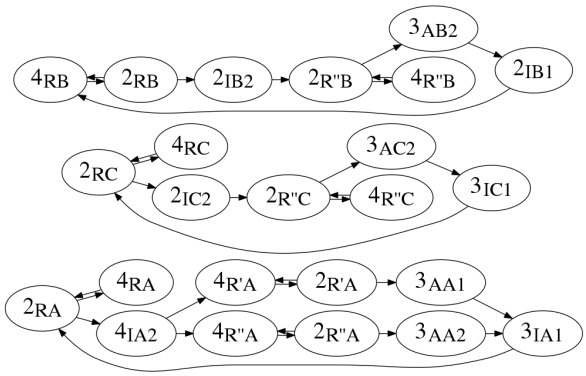


Fig. 7. Some of the full, consistent subgraphs of the dependency graph, generated by the developed tool. Each of these subgraphs represent an infinite set of partial deadlock states.

5. CONCLUSION

This paper describes a tool used to implement a decision procedure for checking the existence of reachable generalized circular waits in parameterized networks. By the nature of parameterized networks, there may be infinitely many such generalized circular waits. In the framework of (Zibaeenejad and Thistle (2017, 2015)), all the generalized circular waits are represented by a finite set of full, consistent subgraphs of the dependency graph of a network; indeed, each full, consistent subgraph can be interpreted as a regular language, each word of which encodes a generalized circular wait. This paper goes beyond this framework by allowing networks to include multiple input processes; the notion of consistency of a dependency subgraph is therefore appropriately generalized.

The developed tool receives a PCN as its input and generates maximal full, consistent subgraphs of the dependency graph by forming 'super-nodes' and 'supergraphs.' Nodes of the supergraph are connected by an edge if the corresponding cycles intersect. These experimental results suggest that the present theoretical results can be usefully extended. The theoretical extension of the framework is the subject of current research.

REFERENCES

Chandy, K.M., Misra, J., and Haas, L.M. (1983). Distributed deadlock detection. *ACM Trans. Comput. Syst.*, 1(2), 144–156.

Emerson, E.A. and Kahlon, V. (2000). Reducing model checking of the many to the few. In *In 17th International Conference on Automated Deduction*, 236–255.

Emerson, E.A. and Kahlon, V. (2002). Model checking large-scale and parameterized resource allocation systems. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS '02, 251–265. Springer-Verlag, London, UK.

Emerson, E.A. and Kahlon, V. (2004). Parameterized model checking of ring-based message passing systems. In *In Proc. of CSÂŠ04, volume 3210 of LNCS*. Springer.

Li, Y. and Wonham, W.M. (1993). Control of vector discrete-event systems. i. the base model. *IEEE Transactions on Automatic Control*, 38(8), 1214–1227.

Wonham, W.M. (2012). Lecture notes on supervisory control of discrete event systems. Available at http://www.control.utoronto.ca/cgi-bin/dldes.cgi, [Jan. 30, 2013].

Zibaeenejad, M.H. and Thistle, J.G. (2017). Deadlock analysis of parameterized-chain networks. *IEEE Transactions on Automatic Control*, 62(4), 2064–2070.

Zibaeenejad, M. and Thistle, J. (2015). Dependency graph: An algorithm for analysis of generalized parameterized networks. In *American Control Conference (ACC), 2015*, 696–702.