International Journal of Computational Intelligence in Control, 9(1), 2017

Structural Reasoning: An Approach to the Evaluation of System State Spaces

Andrei Lobov, Jose L. Martinez Lastra

Tampere University of Technology, Department of Production Engineering, P.O. Box 589 33101 Tampere Finland

Abstract: *Formal validation of the industrial systems requires new ways to make this process more user-friendly. The central question of the formal validation is "Did we build the right system?" However, answering of this question requires an understanding of what is the "right system". Instead, a "What we did?" question can be asked at the first place. Provided a set of system attributes (e.g. I/Os), the user can investigate what are the relationships among selected attributes including different possible scenarios. Therefore, following new analysis techniques the applicability of formal methods can be increased. This paper describes a method to evaluate system state spaces called structural reasoning. The system state space or reachability graph is considered as a finite structure that can be manipulated to provide more readable result. The structural reasoner tool, which is also discussed in the paper, simplifies an analysis of the system. It can be seen as an addition to the traditional methods such as a model-checking.*

Keywords: *Structural reasoning, Petri Nets, Data mining, Formal validation.*

1. INTRODUCTION

Programmable Logic Controllers (PLCs) can be seen as the main devices in implementation of deterministic control at the factory floor. Industrial programmable systems are developed and programmed by engineers using either standard (e.g. IEC-61131) or vendor specific programming languages. The latter often resemble the languages of the mentioned standard–IEC-61131–which are ladder diagrams, function block diagrams, instruction list and structured text. A simplified process of system design can involve narrative descriptions discussing the desired behavior that should be coded by the engineer. Often the next step after the narrative descriptions is the direct coding of the behavior as it was understood by the engineer-programmer.

In order to assist the engineer and to ascertain correct system behavior, formal methods based on rigorous mathematical constructs can be employed. The application of formal methods however can be a high-skill demanding process and can take considerable time share of the entire project [1]. Traditionally formal validation of a system sets a question: "Did we build the right system?" This question contains one important aspect for the successful validation–that is, the understanding what is the *right* system. Answering such auxiliary question is not a trivial task. It is often the case that due to problems in communication between the customer ordering a system and the developer implementing it, the understanding of right system is not fully attainable. Furthermore, the customer as human being sometimes may not fully understand what s/he needs. Taking into account that as the time goes the understanding of the *right* system may also alter–the problem of answering the traditional question of the formal validation becomes even more complicated.

This article discusses an approach for reasoning on system correctness. It was developed based on well-known Venn diagrams [4] and more specifically on Edwards-Venn diagrams [2] introduced in 80s of the last century. The diagrams serve here as a visualization tool. Having system in hands, the developer can generate a system state space (reachability graph) showing all possible states system can reach. Traditionally, the state spaces can be investigated by means of model-checking [13], where certain paths in state space can be tested. In the method presented here, the state space is considered as a large population of nodes where each node has a unique set of attributes and in addition it may share some attributes with other node(s). Based on the structure of the state space and attributes distribution, a statistical approach which authors called 'structural reasoning' is applied to infer the properties of the whole system. The structural reasoning in comparison to the traditional methods allows to focus on the dependences between the selected attributes rather than going relation-by-relation among attributes. The latter case can be found in model-checking: in order to check something in the state space, the engineer need to envision that 'something' and express it in the corresponding language. Therefore, each time it is required to check a formula to hold in state space it is required to envision it first. The structural reasoning on the other hand can provide with hypotheses on the relations themselves which can be found in the system among the attributes. Therefore, the result of the structural

reasoning is a spectrum of possible relations among the attributes. In other words the answer to the question "what we did?"

The structural reasoning is demonstrated on a piece of the equipment that can be found in pallet-based assembly systems used in electronics production – a lifter. The article is organized as follows: second chapter discusses Edwards-Venn diagrams and Timed Net Condition/Event Systems (TNCES)–Petri Nets-based formalism applied here for system modeling. Third chapter describes the principles of the structural reasoning. Fourth chapter discusses an application of structural reasoning to the pallet lifter. Conclusions are drawn in the fifth chapter.

2. EDWARDS-VENN DIAGRAMS AND TIMED NET CONDITION/EVENT SYSTEMS

This chapter discusses the mathematical apparatus used for systems modeling and analysis.

Edwards-Venn Diagrams

British philosopher and the logician, John Venn, has introduced the Venn diagrams to represent the relationships between multiple sets [4, 5]. Nowadays, the Venn diagrams are one of the basic tools used in the schools to describe the sets and relationships among those. An example of Venn diagram is shown in Figure 1. Three sets or concepts are introduced in the diagram: robots, vehicles, and manufacturing machinery. Different intersections of the concepts can be used for classification of particular equipments.

For instance, the intersection of the Vehicle and Manufacturing Machinery concepts can be used to classify different kinds of Automated Guided Vehicles (AGVs) used at the factory floor and also involved in some basic manufacturing processes. Furthermore, the intersection of all three concepts may identify mobile robots applied in a factory, etc.

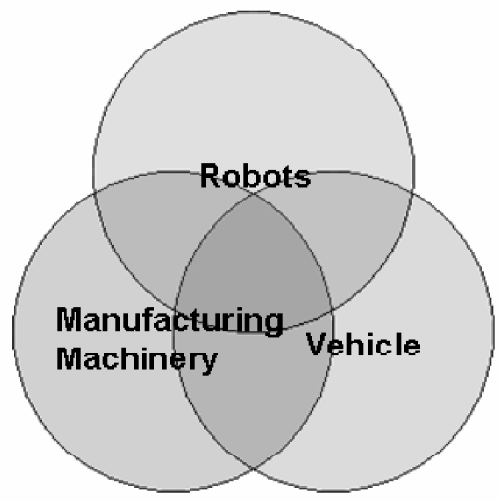

Figure 1: Venn Diagram for Three Sets

Known more than for one century, Venn diagrams was an important aid in the reasoning on rather simple problems involving few factors or sets in the problem domain. One of the reasons for this simplicity of the problems addressed by Venn diagrams is the fact that their visualization is generally limited by four sets. That is, only four sets can be drawn in the Venn diagrams using a general approach where each set is depicted as an ellipse. Until the last quarter of twentieth century, a general solution for the problem of representing Venn diagrams for arbitrary number of sets was unknown. Finally, one possible solution named Edwards-Venn diagrams was given in [2]. Later, [3] provides a state of the art on graphical representation for (arbitrary) many sets listing the main contributions and possible interpretations of the diagrams.

The Edwards-Venn (EV) diagrams allow representing arbitrary many sets and all possible combinations for those. They overcome the limitations of Venn diagrams–the maximum of four sets. An example of EV diagram for six sets is shown in Figure 2.

The way to draw EV diagrams can be expressed as follows. The first three sets can be represented as basic shapes such as circle and rectangles. Although it does not really matter for EV diagram which shape represents what set, for the explanatory reasons lets assume that the circle is the "first" set, the rectangle with the longer width than height is a representation of the "second" set and, finally, the "third" set is represented by the rectangle having its width smaller than height. So far an EV diagram having just these three sets has no difference compare to the *traditional* Venn diagrams. The difference comes with the addition of the fourth set. Starting from the fourth set the boundaries of Venn areas in EV diagram are represented as closed curves following an equation:

$$y = r \cdot \cos(2^i \cdot x) / 2^i + r \qquad (1.)$$

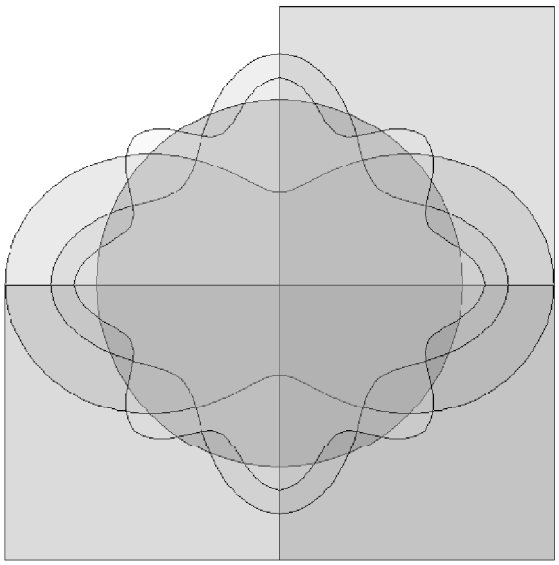

Figure 2: Edwards-Venn Diagram for Six Sets

where

- y–is the radius in the polar coordinate system;
- x–is the angle in the polar coordinate system;
- r–is the radius of the drawn Venn area;
- i–is the curve's number to be drawn $i \in [1, N\text{-}2]$. Therefore, the first curve represents the "fourth" set and the last (N-2) curve stands for the "last" (N-th) set.

Once the curves are obtained, the polar coordinates x and y are translated to the Cartesian coordinates applying the following transformations:

$$X = |y| \cdot \cos(x) + X0 \quad (2.)$$

$$Y = |y| \cdot \sin(x) + Y0 \quad (3.)$$

where

- X–is the abscissa in the planar Cartesian space;
- Y–is the ordinate in the planar Cartesian space;
- $X0$, $Y0$–is the centre of the Venn area;
- x, y–are the polar coordinates according to the equation (1).

This is a bit simplified scenario for the creation of EV diagram. In the ideal case, the curves should be drawn on the spherical surface, and then the stereographic projection has to be applied to derive the diagram. The difference between the "simplified" method presented here and the "complete" one using the stereographic projection is that the former converges faster to the circle ("first set") as the newly added sets oscillate around it. However, firstly, the number of studied sets has to be large to pose the problem for the diagram application as a visual aid, and secondly, for the implementation in a computer allowing simple zoom out/in functions along with configurable radius ('r') parameter, the problem can be easily overcome. Thus, each unique region of Edwards-Venn diagram can be identified in a dedicated tool.

What information can be retrieved from the EV diagrams? First of all, it should be noticed that the neighboring regions in EV diagram differ by one concept only. That is, if the diagram shown in Figure 2 classifies the elements according to six given concepts, the elements of the neighboring regions differ from each other by one concept only. It corresponds to so called Gray code (e.g. {000, 001, 011, 010, 110, 111, 101, 100} in a case of three concepts). Furthermore, if the sine function is to be used instead of cosine in equation (1.), the resulting diagram corresponds to the natural order or the binary-counting order for the concepts, i.e. {000, 001, 010, 011, …, 111} in a case of three concepts.

How this information can be used in relation to the validation of industrial systems behavior? The ordering of the concepts or the attributes as they are called in the Structural Reasoner tool may assist in classification of system evolution principles. The Gray code may be seen as the steady advancing system evolution through the selected attributes, while the binary-ordering may assist in revealing more complex patterns in system behavior.

Timed Net Condition/Event Systems

Net Condition/Event Systems were introduced in 1995 [14]. These are the Petri Nets derived formalism. Later the formalism has been extended to express timing [15]. Figure 3 shows an example of Timed Net Condition/Event Systems (TNCES) module.

TNCES is a typed formalism. That is, newly developed modules get a unique type assigned that gives a possibility for modules reuse in the models–there can be several modules of the same type differentiated by the unique names. Besides the places, transitions and flow arcs that can be found in the ordinary Petri Nets (e.g. {$p1$, $p2$}, {$t1$, $t2$} and {($p1$, $t2$), ($t2$, $p2$), ($p2$, $t1$), ($t1$, $p1$)} in Figure 3), the TNCES is also composed of a set of event arcs (e.g. {(turn_on, $t1$), (turn_off, $t2$), ($t2$, turned_OFF)} in Figure 3); set of the condition arcs (e.g. {(enabled, $t1$), ($p1$, act_ON), ($p2$, act_OFF)} in Figure 3). The modular representation of TNCES allows clear definition of the interfaces by means of event and condition I/Os. In Figure 3, the set of event inputs is {turn_on, turn_off}; the set of condition inputs {enabled}, the set of event outputs {turned_OFF}; and the set of condition outputs {act_ON, act_OFF}. A timing interval can be assigned to the flow arcs connecting places and transitions. Once the marking of the place changes and becomes greater than zero, the place clock is started. The corresponding post-transition becomes enabled by time if the lower bound of the timing interval (if defined) is reached by the place clock. The transition should fire before the upper bound of the timing interval is reached. Otherwise, the transition becomes disabled. In the given example, ($p1$, $t2$) flow arc has a timing interval assigned, meaning that the token should reside at least for 10 time units at place $p1$ and at most 30 time units–otherwise transition $t2$ becomes disabled for good.

Due to condition and event arcs, the semantics of the net is extended. Condition arcs originate from a place and

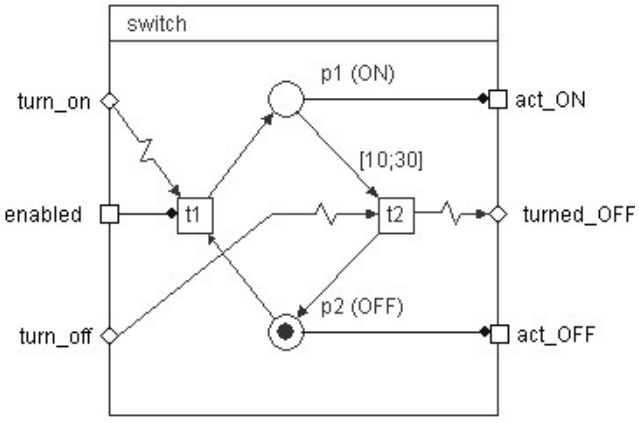

Figure 3: A TNCES Module

lead to the transition(s). These carry the "state information". Having the place marked can cause the firing of the transition(s) in another part of the net. Event arcs on the other hand interconnect transition(s) to transition(s). These signal a change in one part to another part of the net.

TNCES is used here to model industrial programmable systems. The program developed in some industrial language can be translated to the TNCES. The state space is then constructed, where each state is a unique marking of the TNCES modules. TNCES modules in the model of a system represent different parts and aspects of the system, e.g. I/O modules (similar to Figure 3), control logic, and controlled processes [8]-[12]. The resulting state space can be supplied for model-checking and used in structural reasoning.

3. STRUCTURAL REASONING

The structural reasoning was firstly introduced in [16]. Here, its concepts are further elaborated and demonstrated on industrial lifter example. Application of the one of the traditional methods in formal validation such as model-checking [13] allows to test complex scenarios if these hold in the reachable state space (reachability graph) of the system. A Computational Tree Logic (CTL) is one of the languages allowing to express the test scenarios, which are usually obtained from the narrative descriptions (requirements) of the system. Therefore, an engineer at the factory floor needs first to understand the requirements and than express these in the given formal language. The CTL formula representing some tested system scenario can take a complex form as it exemplified in Figure 4.

The formula is given here for the illustrative purposes. One should be familiar with the CTL syntax to interpret it. But knowing the syntax is not enough. Additional information is required to interpret such a formula: the meaning for the markings of certain places ('$m(x) = n$') has to be known as well. Such problems are common for any situation where a higher-level or dedicated language has to be used to translate some aspects from one domain to another, so that the solution can be obtained in that 'another' domain and then the result is again "translated" back to the source domain.

The structural reasoning can be applied to ease the tasks of moving between the different domains. The Structural Reasoner (SR) tool, a part of MOVIDA Framework, implements this possible solution.

The idea behind the SR is to analyze the network of nodes having some properties assigned to them. Using the Venn or Edwards-Venn diagrams and the attributes graph derived from those, one can analyze the behavior patterns of the system and identify interrelationship between the attributes. The SR operates with the network *Net*, which can be defined as a tuple:

$$Net = \{N, C, A, \mu\} \tag{4.}$$

where,

- N is a finite set of the nodes;
- $C \subseteq (N \times N)$ is a finite set of the connections between the nodes;
- A is a finite set of attributes; finally,
- $\mu: A \to N$, is a map between the attributes and the nodes.

System state space S (a connected directed graph) can be represented as follows:

$$S = \{St, Tr, M, \varphi\} \tag{5.}$$

where,

- St is a finite set of states;
- $Tr \subseteq (St \times St)$ is a finite set of transitions;
- M is a finite set of TNCES markings;
- $\varphi: M \to St$ is a function mapping one marking to each state.

The SR tool can be used to analyze the state space as a network. The analysis method is suitable not only for the state space, although this is the main domain in the current discussion. In general, the SR can be applied to any phenomena that can be described similarly to the equations (4.) and (5.). That is, some structure or network having its nodes interconnected and some attributes assigned to the nodes. The network should not be necessarily restricted to a connected graph, as it is the case with the state space.

Further, the attributes graph can be generated for the network. The attributes graph *AG* is defined as follows:

$$AG = \{N_a, C_a, A, \mu_a\} \tag{6.}$$

where,

- N_a is the a finite set of attribute nodes labeled with the number of nodes from N (4.) which they represent,
- $C_a \subseteq (N_a \times N_a)$ is a finite set of the connections in the attributes graph labeled with the number of possible connections between N_a (based on C (4.));
- A is the same as in (4.);
- $\mu_a: A \to N_a$, is a map between the attributes and the nodes in the attributes graph.

The difference between equations (4.) and (6.) is that the former represents the entire net (the state space), while the latter is a compact representation of the network obtained according to the selected attributes. In comparison to the network *Net* representing the state space of the system, its attributes graph *AG* may be unconnected.

What information attributes graph (*AG*) can provide [16]?

- First of all it highlights all the possible scenarios in a system involving selected attributes. The possible scenarios are obtained as a result of the SR. In the case

E[EF(m(p300) = 1 and m(p205) = 1 and m(p521) = 1 and m(p471) = 1) U AF(m(p300) = 1 and m(p204) = 1 and m(p521) = 1 and m(p471) = 1)]

Figure 4: An example of CTL Formula

of model-checking and usage of CTL, each of the possible scenarios has to be first envisioned by the engineer to create the formula and validate it.

- Secondly, the attributes graph may have unconnected nodes. Labeled with zero, those represent the situations that can never occur in a system. Checking the same by means of model-checking would require dedicated formulae to be developed.
- Thirdly, the labels for *AG* nodes can provide information on the number of nodes/states associated with the property. In the context of the state space, it may reflect the 'balance' between different states in the system.
- Fourthly, the number of outgoing connections may provide the link to the 'probability' of going from the states where one property holds to the states where another property holds.
- Fifthly, the number of self-connections can be used to analyze the connection degree for the states. If the number of self-connections divided by the number of the network nodes in the *AG* node is greater than 1, there are several possible paths through the region of a state space represented by the *AG* node.

The SR can be used in identification of the key attributes of the system. The key attributes are the ones that provide a full coverage of the state space. That is, the entire state space can be expressed in terms of the given attributes. For example, consider traffic lights with a simple control changing the lights between the 'red', 'yellow' and 'green' colors. Besides the outputs of the controller that are connected to the corresponding color section of the traffic lights, the controller program may contains another variables that all together make the condition for the light to change from one to another. It would be natural to suppose that the variables representing the lights will serve as the key attributes of the program in relation to which all the rest of variables are defined.

Finding the minimal set of the attributes can be an important task in system analysis. Knowing the key attributes, one can use these as a basis for studying the behavior of other attributes of the system. The key attributes should provide the full coverage of the state space. That is, the entire state space can be expressed in terms of key attributes (e.g. for the traffic lights the reasoning can look as follows: the region of the state space, where the red light is ON, is followed by the region with yellow light ON, etc.)

The density ($Dens_a$) of an attribute i (A_i) in a given state space (S) is defined as follows:

$$Dens_a = |A_i| / |S| \qquad (7.)$$

where

- $|A_i|$, is a number of states in the state space where the given attribute holds;
- $|S|$ is a size of the state space.

The key attributes all together make a density to be equal to '1':

$$Dens = (|A_i| + |A_k| + \ldots + |A_z|) / |S| = 1 \qquad (8.)$$

where $i, k, \ldots, z$ form a set of the key attributes. An example of an attribute can be a variable – a sensor, or an actuator in the control program. Also composite attributes can be considered, for instance when certain combination of variables holds at the same time. In terms of EV diagram discussed before, such attributes are represented by different regions of the diagram. The regions in EV diagram form the necklaces [3]. That is, the regions representing different number of intersecting sets (in our case–states sharing a number of attributes) that radiate around the center of diagram (forming a loop–necklace–around the center). Figure 5 highlights different necklaces for the same diagram shown in Figure 2. Starting with the black color (Figure 5), six regions located outmost form the center of the diagram are highlighted to represent the first necklace that may contain 'pure' elements that belong to one and only one of the given sets (contain only one preselected attribute). Then, using white and black colors in turn, the following necklaces are highlighted. The second necklace is represented with 15 regions (a number of unique ways to select two sets out of six), etc. The last necklace contains only one region–an intersection of all six sets (the states representing this region must contain all the six attributes).

The necklaces can serve in analysis of system behavior. They can show what is the maximum number of the attributes required to fully cover the state space in search for key attributes.

4. CASE STUDY: A LIFTER

The electronics production is one of the application domains for pallet-based assembly systems. A pallet holding a product or its components travels through the system and gets

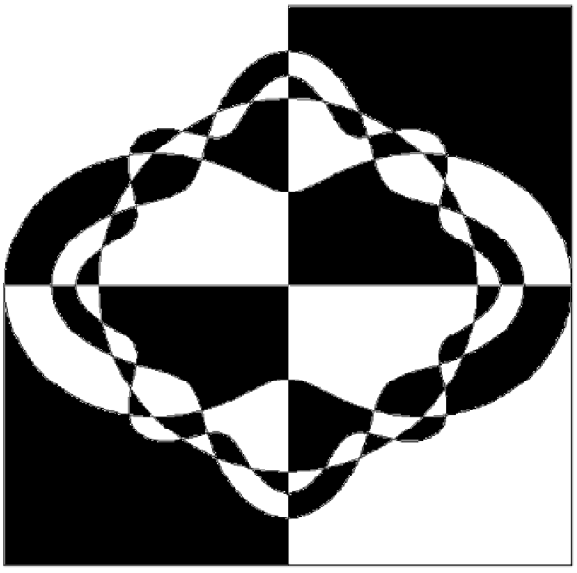

Figure 5: Necklaces of Edwards-Venn Diagram

assembled while visiting different manual or robotic workstations. The conveyor systems transporting pallets may contain two levels in order to allow pallets circulation in the system. Figure 6 shows lifter equipment attached to the mainline conveyor segment. The lifter can receive a pallet at one level and transport it to another level of the conveyor system. Having a line composed of the workstations and mainline, one could add start and end lifters at the beginning and at the end of the main line. The mainline should be a two level conveyor. Therefore at the end lifter the pallet is received at the upper level and transported to the lower level, where the pallet is transported to the start lifter which raises it to the upper level of the mainline. Such an installation allows pallet circulation while particular workstations are busy processing some other pallets. Therefore the blockages are avoided since pallets do not have to wait for the station to get free. Instead, the pallet can come to the same place at the next cycle of the main line.

Figure 7 shows a blueprint of the start lifter. The lifter receives a pallet at the lower level and transports it to the upper level. The labels in the figure denote different sensors and actuators. Each conveyor segment has a sensor (B1/2/3) and a motor (M1/2/3). While receiving a pallet, the motor runs to move the belt to load a pallet on the sensor. The lifter is composed of three segments–lower terminal (that receives a pallet), sledge (a conveyor segment that can also be moved vertically between the terminals), and upper terminal (used to unload the pallet to the mainline). Auxiliary sensors and controls are used to select different modes of the lifter (S1-S5) and to provide safety restrictions on the vertical motion of the lifter (S7 and S8). The sensor B5 ensures that there is no any obstacle between the terminals and the sledge, which serves as a condition for allowing the vertical motion.

Using this information, a programmer takes a controller device and writes a program to implement specified behavior. This particular system was programmed using ladder logic

Figure 6: Lifter Installation in the Assembly Line

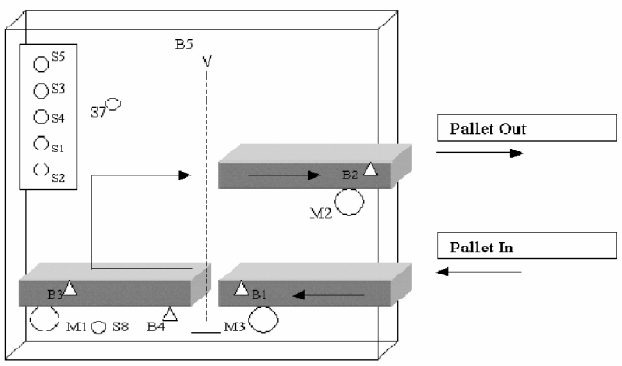

Figure 7: A Blueprint of a Lifter

for Omron™ CPM1A PLC. Then the program was translated to TNCES to be formally validated. An example for the CTL formula for the lifter was given in Figure 4. This formula expresses a scenario when the pallet is not fully loaded to the sledge and the lifter starts vertical motion – which should be forbidden. In order to develop such a formula the engineer must first understand the requirements, envision such a situation and not forget to test it. Traditional testing as well as simulation have one drawback–they lack clear stopping criteria. That is, it is impossible to say if there were enough test runs to ascertain valid system behavior.

In structural reasoning the engineer selects the attributes (sensors and actuators), which is easier for his/her understanding, and tries to reveal interdependences among these attributes. The system state space *S* (5.) for the lifter contains 20007 states computed in MOVIDA Framework.

For system state space analysis, the data should be selected first, namely the attributes to be investigated. Figure 8 depicts the process of selecting attributes (a). Once the attributes are selected, the structural reasoner tool can be invoked. In the structural reasoner, one can select the subset of attributes to study (Figure 8–b). The attributes can be used to generate the EV diagram, to compute the attributes graph and density function.

Figure 9 depicts the attributes graph (*AG*) obtained from the 20007 states of the reachability graph of the system for the selected attributes. There are 31 nodes in total in the *AG*. Each node has a unique color received from the corresponding region of EV diagram which it represents. The nodes contain the number labels representing the number of states that belong to the corresponding EV region–share the same set of attributes. Arcs interconnecting the nodes having a perpendicular bar on them with a number label represent an amount of possible connections between the regions of the state space. In general, the numbers in the nodes should sum up to the size of the state space (i.e. 20007) and the connections to the number of connections between the states in the state space. The *AG* can be seen as a compact representation of the state space. At the fist glance to the *AG* by moving the mouse cursor from node to node, it is possible to observe from the appearing tooltips that it is

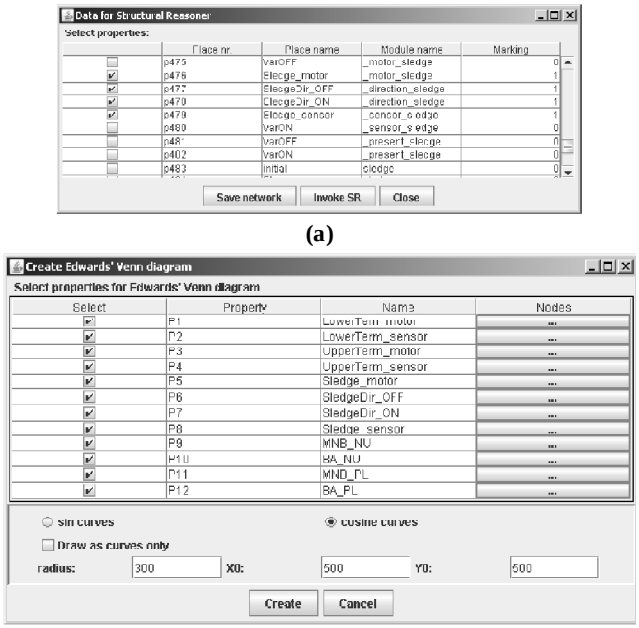

Figure 8: Preparing Data for Structural Reasoning (a) and Selecting the Attributes for Edwards-Venn Diagram Generation

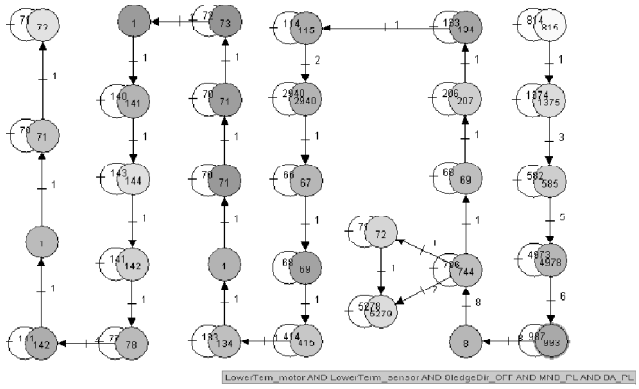

Figure 9: Attributes Graph (AG) for the Selected Attributes

possible for a pallet to travel through the lifter and be unloaded from the upper terminal. This can be concluded by observing the interplay of the motors and sensors of the conveyor segments–starting at the lower terminal and coming to the upper terminal sensor, which will be eventually activated, and after that the pallet leaves the lifter. This can be considered as a normal behavior. However, the *AG* gives another important aspect to consider. While the system is in the node labeled with '744' (there is only one such node in the *AG*), the system may go to deadlock node labeled with '5279'. As can be analyzed by checking the meaning of the nodes (looking at what attributes are assigned to them), it appears that there is such a scenario possible when the pallet is stuck at the lower terminal. Usually such cases are hard to envision and test. What is important for the given example is that no any motors or actuators are activated once such situation is met. In the factory floor, the operator needs to intervene to correct this case. Such situation became possible due to TNCES plant model interconnected with TNCES controller model allows such a scenario. And as a result, it can be validated by means of the structural reasoning on the entire system state space.

Furthermore, one can indentify the key attributes while generating the density function for the necklaces. Figure 10 shows the density distribution for the *necklaces*. Since there are 12 attributes were selected, there are 13 (12 + 1 (Ø)) necklaces available. For example, 0.3 for the fourth necklace grouping four attributes in the graph means that 30% of states in the state space simultaneously contain 4 preselected attributes. It is worth to check if the key attributes can be found to simplify the analysis. In terms of density distribution, the function should be maximized by removing some of attributes in order to check if these are already covered by the necklaces of smaller order (farther from the center). The search for the key attributes gives a result shown in Figure 11. There are two attributes that can fully cover the entire state space–these are directions of the sledge conveyor (SledgeDir_ON and SledgeDir_OFF variables (Figure 8)). Although such outcome was expected in this case, the engineer using these key attributes may start gradually adding other attributes to look how the system behave. Eventually the graph shown in Figure 9 or its smaller representation can be obtained in this process.

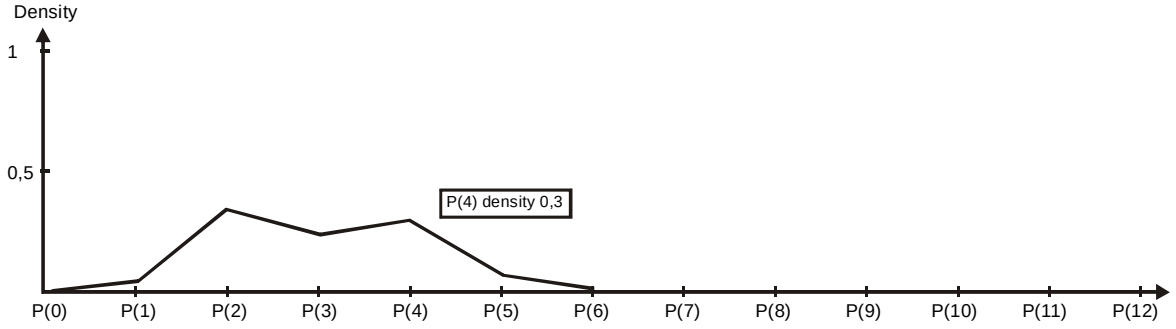

Figure 10: Density Distribution for the Selected Attributes

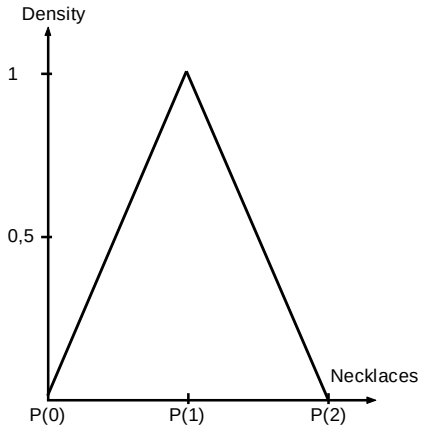

Figure 11: Density Function for the Key Attributes

5. CONCLUSIONS

A novel approach providing a solid view to the structure of nodes having the attributes assigned to them (e.g. system state space) and, therefore, simplifying the system analysis was discussed in the article. The approach shifts the focus of the formal validation from the attempts of answering "Did we build the right system?" question to "What we did?" question. It allows user to reveal all the scenarios for selected attributes rather than using scenario-by-scenario approach.

In a future work, connections to the other theories especially to the statistical methods have to be further elaborated. In general, system state spaces can be treated with some precautions as a large collection of data that can be suitable for data mining where each state is seen as an "individual" having some unique attributes in a large "population" of states.

REFERENCES

[1] P. E. Ross, "The Exterminators", *IEEE Spectrum*, **42**, (2005), 36-41.

[2] A. W. F. Edwards, "Venn Diagrams for Many Sets", *New Scientist*, **1646**, (1989), 51-56.

[3] A. W. F. Edwards, "Cogwheels of the Mind: the Story of Venn Diagrams", *The Johns Hopkins University Press*, 2004.

[4] J. Venn, "On the Diagrammatic and Mechanical Representation of Propositions and Reasonings", *London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, **9**, (1880).

[5] J. Venn, "Symbolic Logic", *London: Macmillan*, (1881).

[6] G. Frey and L. Litz, "Formal Methods in PLC Programming", *In Proc. of IEEE Conference on System Man Cybernetics*, (2000), 2431-2436.

[7] A. Lobov and J. L. Martinez Lastra, "Data Mining of Systems State Spaces", *In Proc. of 2nd International Conference on Changeable, Agile, Reconfigurable and Virtual Production*, July 2007.

[8] A. Lobov, C. Popescu and J. L. Martinez Lastra, "On Reachable State Space Reduction for Formal Validation of Scan-based Systems", *In Proc. of 4th IEEE International Conference on Industrial Informatics*, August 2006, 79-84.

[9] A. Lobov, J. L. Martinez Lastra and R. Tuokko, "Application of UML in Plant Modeling for Model-based Verification: UML translation to TNCES", *In Proc. of 2nd IEEE International Conference on Industrial Informatics*, August 2005.

[10] A. Lobov, C. Popescu and Martinez Lastra, J. L., "An Algorithm for Siemens STL representation in TNCES", *In Proc. of 11th IEEE International Conference of Emerging Technologies and Factory Automation*, 641-647, September 2006.

[11] A. Lobov, J. L. Martinez Lastra, R. Tuokko and V. Vyatkin, "Modelling and Verification of PLC-based Systems Programmed with Ladder Diagrams", *In Proc. of 11th IFAC Symposium on Information Control Problems in Manufacturing*, April 2004.

[12] H.-M. Hanisch, A. Lobov, J. L. Martinez Lastra, R. Tuokko and V. Vyatkin, "Formal Validation of Intelligent-Automated Production Systems: Towards Industrial Applications", *International Journal of Manufacturing Technology and Management*, **8**(1-3), 75-106, 2006.

[13] E. M. Clarke, O. Grumberg and D. Peled, "Model Checking", *MIT Press*, 1999.

[14] M. Rausch and H.-M. Hanisch, "Net Condition/ event systems with multiple condition outputs," *In Proc. of Symposium on Emerging Technologies and Factory Automation*, Paris, France, **1**, 592-600, October 1995.

[15] H.-M. Hanisch, J. Thieme, A. Lüder, O. Wienhold, "Modeling of PLC Behavior by Means of Timed Net Condition/Event Systems", *In Proc. of 6th IEEE Conf. on Emerging Technologies and Factory Automation* (ETFA'97), Los Angeles, (1997), 391-396.

[16] A. Lobov, J. L. Martinez Lastra, "Structural Reasoning in Proving System Correctness", *In Proc. of IEEE 12th International Conf. on Emerging Technologies and Factory Automation* (ETFA' 2007), Patras, Greece, (2007), 681-688.