Formal Verification of Analysis Approach for Enterprise Information Systems Architecture Using Hypergraph Representation Based on Finite State Machines for Supporting Business Process Requirements

Khawla Bouafia
Eötvös Loránd University

Bálint Molnár
Eötvös Loránd University

Enterprise information systems (IS) should align processes in organizations to ease strategies success and solve problems using different approaches. Modelling is a way to represent processes and supporting enterprise architecture (EA) which should comply with a set of rules and constraints.

Model-checking becomes a major area of research which used for formal verification of various properties translated into mathematical logic. In this paper, an approach of model checking on hypergraph representation based on a finite state machine(FSM) which supports the alignment of business process(BP) requirements will be presented to check its correctness and the satisfaction of some properties which need the checking.

Keywords: Information System (IS), model checking, Linear Temporal Logic (LTL), property, hypergraph representation, Finite State Machine (FSM)

INTRODUCTION

EA is a capability to envision, plan, design, lead, manage, and control enterprises, systems, and/or processes in current, transitionary, and future states, and defines the relationships between them. It describes an organization in terms of its strategy, structure, information flows, value streams, as well as its business. EA enables the business to understand its current composition, utility, costs, and sources of value generation.

With EA envisioning and developing the programs and projects necessary to support realization of the organization's future strategic objectives. It improves the quality and performance of BP and enhances productivity across the organization by unifying and integrating data linkages, that's whey during IS (process) development, we can find such kinds of errors in the implementation step, caused before i.e., here we pose the question: how can we ensure and solve these various problems before the implementation and ensure the business requirements ?

Diverse logical methods and verification approaches are used to confirm the validation m models and IS processes management, the models can be represented with the semantics and verify it formally with its debug grammatical errors using formal methods like model checking.

To answer the question posed before, we propose to utilize formal method to prove that the model of the system that wants to be implemented is verifying some critical requirements, in our case, we will propose to apply model checking on the model proposed in our previous work (Khawla, 2018) which support BP, the hypergraph defined with logical expressions and mapped in a core of FSM that formal model which can be checked easily using applying Linear Temporal Logic (LTL) formulas to make sure its correctness.

This paper is structured as follows, after the introduction in the first section, we will present different related and recent works in the same domain, next in the third section presents a background about formal methods, various objectives of formal verification, model checking and its techniques. After that, basics of the model used introduced in section four, then the method applied with LTL formulas of the different properties that supports BP requirements with hypergraph based on FSM will be described in section five, finally we will conclude our idea and future work in the conclusion section.

RELATED WORKS

The IS and processes are becoming more and more flexible and sometimes instead of recording the system (Bouafia, 2019) it typically suffices to reconfigure the system on the basis of a process model, at various stages of the system development, The main aim of model checking is to provide to decision procedure efficiently for evaluation of several logical formulae (Gottlob, 2004) over finite relations structures, the hypergraph are the best example of such structure especially because we combine it with FSM which rich it with more properties that can be verified.

We can use model checking techniques (Emerson, 2008) to verify the specification. In various domains of computer science, in both hardware and software (Chou, 2012) (Jindal, 2014) sides researches in (Morimoto, 2008) describe and give a general idea about checking process without mentioning variability and compliance of using formal verification. However, the overview (Groefsema, 2008) study the offer goals of formal verification, techniques, and frameworks for process modeling.

Here is a plethora of studies and publications in the field of model checking methods and applications on various models, it is possible to check a real-life process specification expressed by Petri nets (Gomes, 2005) or semi-form Unified Modeling Language (UML) activity diagrams (Kherbouche, 2019) (Grobelna, 2014).

Several researchers have addressed the problems related to workflow change (Van der, 2001) (Rinderle, 2004) In the first phases of design models, most errors should be spotted to correct it as early as possible because of the costs during the system life and running (Barry, 1981). Model-checking is one of the most know detectors in behavioral designs (Biere, 1999), the translation into the language of automated tools from the rules and state machine mentioned in (Karimi, 2014) called model checkers which offer support for policy analysis and allow checking if a policy (set) satisfies certain properties.

Application of graphical representation of algorithm is a very conformable method of specification of dedicated binary controllers. In the case of Petri nets is one of the most adequate methods for the formal design of such controllers. It gives an easy way for representation of concurrent processes and additionally there could be applied mathematical algorithms for formal analysis and verification of the designed model. There are also several algorithms of direct (Tkacz, 2012) or distributed (Wiśniewska, 2012) synthesis of Petri net model into programmable devices (Wiśniewski, 2014).

Besides petri nets, a system may also be modelled as FSM a directed graph of nodes and edges (Van der, 1998) with nodes representing a state system and edges representing a change in state. The authors in (Grobelna, 2016) propose using formal methods and double model checking on Petri nets to ensure correct functionality of the designed distributed logic controller, as a case example they propose a smart home system.

Preliminaries

Before starting our approach and application of the model checking we should know the different principals, concepts and methods used, as well as the several formal methods that exist.

Formal Methods

Software engineering provides various tools for the development of software such as, facilitating the partitioning of software development. Formal methods (Peled, 2019) are based on mathematical and logical principles and are often accompanied by automatic or human-assisted tools, these ways are utilized to formally specify the requirements from the system. A partition of formal methods is as follows:

- **Modeling**: getting an abstract away some details of the real system applying formal ways.
- **Specification**: the first step of the design with the help of such formalism like logic, process algebra, and automata.
- **Testing**: checking the executing (or simulating) the code.
- **Verification:** proving the software correctness with their formal specification.

In recent years, a lot of progress are seen in both, performance of the different techniques and tools in accepting the use of formal methods by the software and hardware industry and expect further new and exciting ideas. We use formal methods for:

- Check the conformance of software conforms with the specifications.
- Testing the process quality to produce.
- Products and block of the issues, defects in the application or the product and quality assurance.
- The verification aims us to solve some properties: Connectivity (determine the process interaction and which are satisfied with nonfunctional requirements); Correctness (the system satisfy to either safety ("*nothing bad*" will happen, ever, during the execution of a system) neither liveness ("*something good* "will happen, eventually, when the system executed); Compensation (how many services can process models implement); Compatibility: it should bridge the gap between BP design and implementation of enterprise IS and satisfy compatibility between business participants in the collaboration.

Model Checking

Model checking is one of verifying approach which gives system model with a specification of interest to including ensuring basic correctness of processes, business compliance checking and variability. It looks like an automated error test or simulation; the manner is useful for detecting any property infringement (i.e; bugs or errors) (Baier, 2008). This formal method used for the checking finite-state systems, given a system model and such specification as a set of formal properties, the model checker verifies if the model meets with the specification or not and ability of generating the counterexample (Debbi, 2018) (see Fig. 1) when the model falsifies the specification (the major advantage of model checker).

Techniques of Specifying and Modeling Systems. The formal verification of a system needs two inputs: one is the given system description and the second corresponding properties will be verified.

The First Input: The Systems Modeling

Generally, the IS describes modules of information processing systems, links among components, design, and analysis principles at the IS level of which the main purpose is to buttress BP (Khawla, 2019). The representation of systems characteristics as e.g. states, configurations through algebraic methods and techniques can be transcribed into graph representations (Molnár, 2014) (Molnár, 2012) FSM, Kripke structures, binary decision diagrams (BDDs), and model extraction from the code will be presented later respectively:

1. **Finite State Machines (FSM):** a standard model used in the mathematical foundation (Molnár, 2017). It can be conceived as an abstract machine having finite set of states (see Fig. 2). with one beginning state only, the change from the current state to another made with transitions which triggered by some event or condition (Qadir, 2014). Basic FSM Patterns (Khawla, 2018)

are: Sequence, Exclusive choice, Multiple choice, Multiple merges, cycle. An FSM defined as a quintuple $(\Sigma, S, S_0, \delta, F)$, where: Σ is the input alphabet; S is a finite set of states (non-empty); The initial state S_0; δ is the state transition function, and the set of final states F, a (possibly empty) subset of final states.

2. **Kripke Structure:** a graph with labeled state transition that can capture the temporal behavior of reactive systems. We can say that the Kripke structure is just a labeled FSM extended to incorporate al labeling function, more formally (Debbi, 2018) it is of a set of states, s, transitions and labels for each states defining property. The structure defined as a 4-tuple $M = (S, S_0, R, L)$ where: S the finite set of states; S_0: initial states; R the transition relation; L is a labeling function that labels every state with the set of atomic propositions that are true in that state.
3. **Binary Decision Diagrams (BDDs):** a quite old efficient technique to representing state transition systems, it is as the algorithmic basis for symbolic model checkers (Bryant, 2018).
4. **Model Extraction from Code:** the checking of code or the implementation code not the model through some automated model extraction approach.

The Second Input: Formal Specification

There are three types of formal specification techniques which are:

1. **Language-based techniques:** Mathematical technique based on predicate calculus approach.
2. **FSM-based techniques:** an extension of programming language to incorporate the representation of state machine and rules (Yuang, 1988). Techniques like extended FSM, petri nets, abstract state machines.
3. **Temporal logical techniques:** which are statements of ordering of events and their actions. There are two important subtypes of LTL and Computation tree logic (CTL).
   (a) Linear Temporal Logic (LTL): a common specification formalism for formal method tools, it is frequently used for specifying properties of reactive and concurrent systems. It describes the allowed executions using temporal operators represented by the syntax below:

$$\phi ::= true|false|p|\neg\phi|(\phi \vee \phi)|(\phi \wedge \phi)|(\phi \rightarrow \phi)|G\phi|R\phi|F\phi|(\phi W \phi) \quad (1)$$

represent fixed properties of the states of the program and the various temporal operators syntax are: [ ] for G (globally); <> for F (future); <> for X (neXt) and U for U (strong Until). A property holds in a model if it holds on every path (Kwiatkowska, 2018) starting from the initial state, LTL is used mostly for applications in software verification.

(b) Computation Tree Logic (CTL): an example of branching temporal logic has path quantifiers such as: A (for all paths∀) and E (there exists ∃ a path) named universal and existential quantification, respectively. It allows branching time and quantifiers, CTL is used mostly for applications in hardware verification

Basic Concepts of the Model

Various detecting defects at the modeling level propose using formal verification techniques to ensure the correctness of the model, in our approach we will try to prove the correctness of model proposed in our previous work (Khawla, 2018) i.e.; check if the mathematical model: hypergraph in a core of an FSM model satisfying with some important properties or not. Each FSM pattern described by a formula which can be verified easily later, in addition to that the highway of presentation of entities, activities, or relations express an easy way to check some important properties. The important basics on the model are:

Definition 1. FSM's elements are represented in hypergraph by hyperedge h_i where $h_i \in E_{FSM}$ the set of hyperedges. The elements of $h_i$are:

Finite set of n states S (vertices): $V = \{v_1, v_2 \dots v_n\}, V_s \subseteq S$

- Finite set of m transitions T (hyperarcs) be belonging to A_T where $\overrightarrow{h_i} \in A_T, T \subseteq A_T = \{\overrightarrow{h_1}, \overrightarrow{h_2} \dots \overrightarrow{h_m}\}$
- The variables or attributes as a label of a transition described by variables belong to an attribute type, $A_{ttr} = \{A_1, T_2 \dots T_j\}$, the set of types $h_i \subseteq V_{i_a} \cup V_{inter_{ba}} \cup V_c$ where $a, b, c, d \in N$

Definition 2. The different important element of the model divided as below:

- The set of vertices $V \subseteq S$ into three sub-sets :V_i, V_{inter} and V_f i.e : initial vertex, intermediate vertices and finals vertices.
- The set of arcs (directed edges) with two subsets: A_{sim}simple arcs and A_{cond} arcs with condition.

NB: The relation between arcs:

$$A_{cond} \cup A_{sim} = A \; and \; A_{cond} \cup A_{sim} = \emptyset \tag{2}$$

This structure represented as nodes embedded into hyperedges and hyperarcs. The hyperarc direction shown the direction of transition where input and output are vertices (the head and tail of a hyperarc) which represent states.

Definition 3. Labelled Hypergraph is a generalized hypergraph that can be extended by some functions and operations: $H = (V, E, label)$ where $|V| < \infty, E \subseteq V \times V, |E| < \infty$ i.e; V, E finite, $\Sigma = \{\sigma \mid \sigma \in \{0,1\} *\}$ set of labels, as binary strings.

The set of data values D can be grasped (efficiency of the representation is left out of the investigation) again as vertices within the hypergraph and it can be interpreted as variables. Over D as the values of sets of variables, sets of operations (OP) can be defined that can be used to describe constraints and rules within formulas. Any FSM's pattern can be represented by hyperedge hi where $h_i \subseteq E_i$, where E_i the set of hyperedges, E can be divided into sub-sets of E where sub-sets of E present a special pattern:

- $\boldsymbol{E_{seq}}$**:** sequence which can use all types of vertices (the tail of the first hyperarc) and use only simple hyperarcs.
- $\boldsymbol{E_{ps}}$**:** parallel split represents a parallel vertex included in the same hyperedge, where various parallel vertices have the same tail from the main vertex.
- $\boldsymbol{E_{ec}}$**:** the exclusive choice pattern represent hyperedge contain all vertices which have the same tail from the main vertex of different hyperarc, we can designate the conditions with values that will be assigned to element of conditions (B*oolean value: true or false*) except null (value).
- $\boldsymbol{E_{mc}}$ **:** describe pattern multiple choice pattern using only two types of vertices V_{inter} and V_f, the express conditions attributes can be used including null.
- $\boldsymbol{E_{mm}}$: present the multiple merges, these patterns is more complicated than to others because of the multiple states merging vertices are included in the same hyperedge it can contain all type vertices and there is no connection merged vertices.
- $\boldsymbol{E_c}$**:** the cycle pattern, a simple structure from hyperedge which presented the head E or vertex V to the tail which can be the same as the head.

We introduced the various important basic parts of the hypergraph model as well as the different patterns of FSM to facilitate the formal verification using model checking approach, next section explains it in detail.

Problem Posed and Verification Method of Model Properties

Design of IS becoming more complicated because of design errors which can result from interleaved access over shared data, process synchronization, specifications dynamic changes and very likely the misunderstanding and misinterpretation by programmers on business logical specifications, for that researchers aim to find solutions (approaches and tools) to investigate general verification techniques for

quality design. Our main goal is to detect the correctness of our model and its satisfying with various properties.

The paper proposes a formal method for verifying our model which base on hypergraph mapped in FSM using model checking, we will study and use a one of verification technique. We will define and verify if some base hypergraph properties satisfy with the model or not? we will apply a verification approach on the hypergraph model because it can be a good example of such structure to help the decision.

The verification must formalize mathematically the process model and verification criteria which should be the defined well because the hypergraph model wants to be verified is in the core of FSM (represented with its semantic), so the model checking is the suitable technique to verify its correctness.

Graphs and hypergraphs are important examples of such structures, our model which is the input based on some concepts are formulated on the base of Kripke structure which is as we mentioned before that is a labeled FSM, so main concepts of the model will be verified using Kripke structure are:

Definition 4. Transition System (Kripke structure), with (v_0) explicit on hypergraph based of FSM. A transition system: $M = h_V, v_0, \rightarrow, L_i$ consists of:

- V finite set of different kind of vertices (V_i, V_{inter} and V_f);
- $\subseteq V$ the initial vertex (v_0);
- $\rightarrow \subseteq v \times v$ transitions between vertices (v_i), (v_{inter}) and (V_f), transition (various edges of the model) can be labeled with an event (actions cited before) a guard and a set of effects that represent actions triggering other effects;
- $V \rightarrow P$ labelling function such that $\forall v_1 \in V, \exists v_2, S_1 \rightarrow v_2 \subseteq V \times V$.

Definition 5. A path π in our system $M = (V, v_0 \rightarrow L_I)$ representing one possible execution of the system that it models, is a (finite or infinite) sequence of vertices and transitions $\pi = v_0\, a_0\, v_1 a_1\, s_1 ..$, where $v_i \in V,\ A_i \in A(s_i)\ and\ \delta(v_i, A_i)(v_i + 1) > 0$ for all i ∈ N.

Such program given as code hardware description language correspond to an FSM i.e; a directed graph consisting of nodes (or vertices) and edges. nodes are system states and edges describe transitions, while the propositions give a property at execution point. Errors in such systems or process can have consequences, hence the urgent need to be able to ensure and guarantee their correctness, Unfortunately, Once the model is complete, properties can be required, and verified. applying to check have focused on finite-state models and either CTL or LTL it is necessary to specify properties on the finite-state abstracted models and this will be our second step:

Formally, the problem can be stated as follows: given a desired property, expressed as a temporal logic formula ϕ, and a structure M with initial vertex v0 decide if $M \models \phi$.

The concept LTL model checking seeks to answer the question (with starting state omitted): Does $M \models \phi$ hold? or, equivalently: $Does\ \forall \pi \in Paths(M).\pi \models 0\ \phi\ hold?\ where(recall)\ \pi \models i\ \phi$ means path at position i satisfies formula ϕ. In this part of the paper, we will give some general properties which can be verified in our model expressed in LTL formulas introduce some general idea of the model or special property of part of the model (pattern) to be sure about its correctness. Propositional variables should be known before to express the set of atomic propositions e.g.: i, f, m. The syntax of the atomic descriptions of a system in LTL expressed by using properties of LTL.

- **Property 1:** starting vertex i must respond to final vertex f. LTL expressed with : $G(S \Rightarrow Ff)$ this property generally express that every process starts with the initial state or initial vertex (v_i) must respond to one a final states or final vertex (v_f), the G express for always (globally) and the F for eventually (in the future). This formula ensures that if a process start so it should finish, i.e; the system is working.
- **Property 2:** When initial action occurs, it will eventually be final action. When intermediate action occurs, it will eventually be in final action LTL expressed with : $G(i \Rightarrow Ff)$ that means globally when the process start from the first vertex it will be at final action (one of the final actions)in the future, i.e.; this property generally express that every process starts with the initial state or initial vertex (v_i) must respond to one of the final states or final vertex (v_f), the

G express G for always (globally) and the F for eventually (in the future).This formula ensure that if a process start so it should finish, i.e.; the system is working

- **Property 3:** initial vertex precedes intermediate vertices (if exists) after one of the final vertices. LTL expressed with: $[\neg q \cup (q \wedge [\neg p \cup s])]$, this property express that globally the the state expressed by q (v_f final state) could not happened only until the state p v_{inter} intermediate state (if it exists in the process) happened and this state too can happened if only the initial state start the process (v_i), this property can express the respecting of the order in the model to guaranty the good sequence of different actions.
- **Property 4**: S exists between Q and R. LTL expressed with : $G(Q \wedge \neg R \Rightarrow (\neg RW(S \wedge \neg R)))$, this property generally express that every intermediate state R in LTL expression should be happened between initial expressed with Q state or initial vertex (v_i) and one of the final states R or final vertices (v_f) the G express G for always (globally) and the W for weak until. This formula ensure the order of the vertices or sates in the model should be respected i.e.; an intermediate (v_{inter}) state should be between the initial state (v_i) and final one(v_f). So, the system is working in respected order.
- **Property 5:** Checks that the model is non-trivial. LTL expressed is: EX true, this property generally express that at least one state success, G for always (globally) and the F for eventually (in the future). This formula ensure that the model is significant.

Generally, two types of properties can be expressed using temporal logic: Safety and Liveness. safety property state that something bad never happens, a simple example of that is the LTL formula $G\neg$ error that means that error never occurs.

Liveness properties state that something good eventually happens, a simple example of that can be expressed better is the CTL formula ($\forall Greq \rightarrow \forall Fgrant$) that means that every request is eventually granted.

CONCLUSION

In this paper, we have presented the formal verification technique which verifies the process model at the design phase of the IS model, these techniques can detect and correct errors of the models as early as possible and in any case before implementation because as we know the correctness of the software models has to be ensured in order to get correct software.

Model-checking verification promises to have an even greater impact on the hardware and software industries, as future work will investigate verification and strategy synthesis techniques for our model where a simulation of this formal verification will be proved soon.

ACKNOWLEDGEMENTS

The project was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002) and by no. ED 18- 1-2019-0030 (Application-specific highly reliable IT solutions) program of the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme funding scheme.

REFERENCES

Baier, C., & Katoen, J.P. (2008). *Principles of model checking*. MIT press.

Barry, B. (1981). Software engineering economics. *New York*, *197*.

Biere, A., Cimatti, A., Clarke, E., & Zhu, Y. (1999). Symbolic model checking without BDDs. In *International conference on tools and algorithms for the construction and analysis of systems* (pp. 193-207). Springer, Berlin, Heidelberg.

Bouafia, K., & Molnár, B. (2019). Analysis approach for enterprise information systems architecture based on hypergraph to aligned business process requirements. *Procedia Computer Science*, *164*, 19-24

Bryant, R.E. (2018). Binary decision diagrams. In *Handbook of model checking* (pp. 191-217). Springer, Cham.

Chou, C.N., Ho, Y.S., Hsieh, C., & Huang, C.Y. (2012). Symbolic model checking on SystemC designs. In *DAC Design Automation Conference 2012* (pp. 327-333). IEEE.

Debbi, H. (2018). Counterexamples in Model Checking-A survey. *Informatica*, *42*(2).

Emerson, E.A. (2008). The beginning of model checking: A personal perspective. In *25 Years of Model Checking* (pp. 27-45). Springer, Berlin, Heidelberg.

Gomes, L., Barros, J.P., & Costa, A. (2005). Structuring mechanisms in Petri net models. In *Design of embedded control systems* (pp. 153-166). Springer, Boston, MA.

Gottlob, G., & Pichler, R. (2004). Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width. *SIAM Journal on Computing*, *33*(2), 351-378.

Groefsema, H., & Bucur, D. (2013). A survey of formal business process verification. *Proc. Int. Sym. Business Modeling and Software Design*. Noordwijkerhout, The Netherlands.

Grobelna, I., Grobelny, M., & Adamski, M. (2014). Model checking of UML activity diagrams in logic controllers design. In *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30–July 4, 2014, Brunów, Poland* (pp. 233-242). Springer, Cham.

Grobelna, I., Wiśniewski, R., Grobelny, M., & Wiśniewska, M. (2016). Design and verification of real-life processes with application of Petri nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *47*(11), 2856-2869.

Jindal, V., & Carr, M. (2014). Model checking of a cash machine system. In *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)* (pp. 258-261). IEEE

Karimi, V.R., Cowan, D.D., & Alencar, P.S. (2014). An approach to correctness of security and operational business policies. *International Journal of Accounting Information Systems*, *15*(4), 323-334.

Khawla, B., & Molnár, B. (2018). An FSM Approach for Hypergraph Extraction Based on Business Process Modeling. In *International Conference on Computer Science and its Applications* (pp. 158-168). Springer, Cham

Khawla, B., & Molnár, B. (2019). *A Survey on Dynamic Business Processes and Dynamic Business Processes Modelling*. The international Conference ICEIS 2019. 556–563

Kherbouche, M., Bouafia, K., & Molnár, B. (2019, December). Transformation of UML State Machine To YAWL. In *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)* (pp. 215-220). IEEE

Kwiatkowska, M., Parker, D., & Wiltsche, C. (2018). PRISM-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives. *International Journal on Software Tools for Technology Transfer*, *20*(2), 195-210.

Molnár, B., Benczúr, A., & Tarcsi, Á. (2012). Formal approach to a web information system based on story algebra. *The European Journal of Applied Economics*, *9*(2), 63-73.

Molnár, B. (2014). Applications of hypergraphs in informatics: a survey and opportunities for research. *Ann. Univ. Sci. Budapest. Sect. Comput*, *42*, 261-282

Molnár, B., & Bouafia, K. (2017). Adaptive Case Management and Dynamic Business Process Modeling A proposal for document-centric and formal approach. *AIS 2017, 12th International Symposium on Applied Informatics and Related Areas*, pp. 3–9.

Morimoto, S. (2008). A survey of formal verification for business process modeling. In *International Conference on Computational Science* (pp. 514-522). Springer, Berlin, Heidelberg.

Peled, D.A. (2019). Formal Methods. In *Handbook of Software Engineering* (pp. 193-222). Springer, Cham.

Qadir, J., & Hasan, O. (2014). Applying formal methods to networking theory, techniques, and applications. *IEEE Communications Surveys & Tutorials*, *17*(1), 256-291.
Rinderle, S., Reichert, M., & Dadam, P. (2004). Correctness criteria for dynamic changes in workflow systems—a survey. *Data & Knowledge Engineering*, *50*(1), 9-34.
Tkacz, J., & Adamski, M. (2012). Macrostate encoding of reconfigurable digital controllers from topological Petri net structure. *Przeglad Elektrotechniczny*, *88*(8), 137-140.
Van der Aalst, W.M. (1998). The application of Petri nets to workflow management. *Journal of circuits, systems, and computers*, *8*(01), 21-66.
Van der Aalst, W.M. (2001). Exterminating the dynamic change bug: A concrete approach to support workflow change. *Information Systems Frontiers*, *3*(3), 297-317.
Wiśniewska, M. (2012). *Application of hypergraphs in decomposition of discrete systems*. University of Zielona Góra Press.
Wiśniewski, R., Stefanowicz, Ł., Bukowiec, A., & Lipiński, J. (2014). Theoretical aspects of Petri nets decomposition based on invariants and hypergraphs. In *Multimedia and ubiquitous engineering* (pp. 371-376). Springer, Berlin, Heidelberg.
Yuang, M.C. (1988). Survey of protocol verification techniques based on finite state machine models. In *Proceedings. Computer Networking Symposium* (pp. 164-172). IEEE.

APPENDIX

FIGURE 1
PRINCIPAL OF MODEL CHECKING

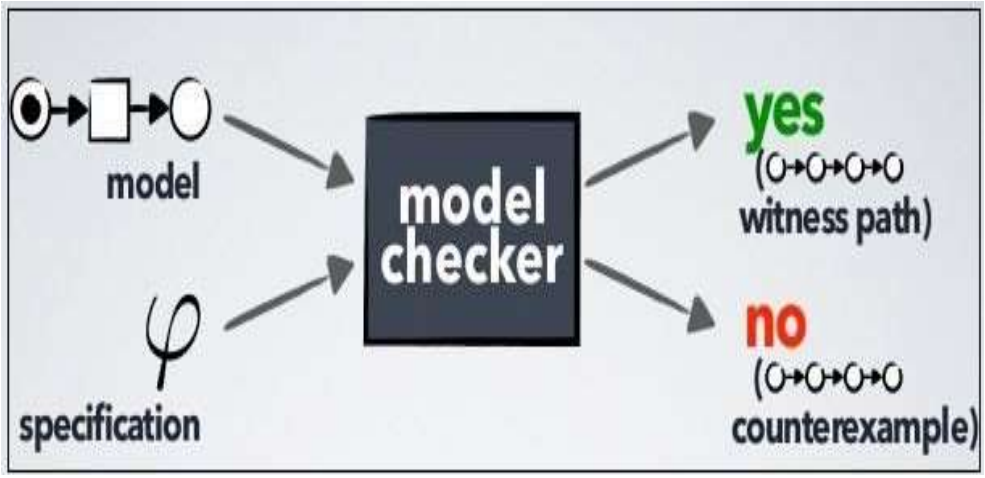

FIGURE 2
FSM ELEMENTS