



HAL
open science

Exploring State Machine CECA Model

Jerzy Chrzęszcz

► **To cite this version:**

Jerzy Chrzęszcz. Exploring State Machine CECA Model. 19th International TRIZ Future Conference (TFC), Oct 2019, Marrakesh, Morocco. pp.388-399, 10.1007/978-3-030-32497-1_31 . hal-02905561

HAL Id: hal-02905561

<https://inria.hal.science/hal-02905561v1>

Submitted on 23 Jul 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Exploring state machine CECA model

Jerzy Chrzęszcz^{1,2} [0000-0003-3257-9726]

¹ Institute of Computer Science, Warsaw University of Technology, Warsaw, 00-665, Poland

² Pentacomp Systemy Informatyczne S.A., Warsaw, 02-222, Poland

jch@ii.pw.edu.pl

Abstract. It was shown during TRIZ Future 2018 conference, that a diagram resulting from Cause-Effect Chains Analysis (CECA) might be transformed into a state machine model. Although the conversion was described with a set of rules, no specific benefits of switching to a state machine approach were presented then.

This paper focuses on enhancing the conversion and exploring the possibilities to simplify the output model without losing its information content. It briefly shows relations between the state machines and the formal grammars, proposes regular expressions as a compressed representation of the processes producing target disadvantages and provides SWOT-like analysis of the behavioral state machine CECA model with respect to the classic structural CECA model. It also shows the similarity of the state machine model to hardware-software approach.

Keywords: TRIZ, Cause-Effect Chains Analysis, state machine model, formal grammar, regular expression.

1 CECA and logical model

Cause-Effect Chains Analysis method was developed by GEN3 in 1990s for building causality diagrams in a systematic way [1, 2, 3]. It is intended to be used after exploring structure and operation of the analyzed system with other TRIZ tools, such as Function Analysis or Flow Analysis. The procedure starts with selecting *target disadvantages* that should be eliminated and then their preceding causes (*intermediate disadvantages*) are investigated one by one until the *root causes* are found, which are recognized as remaining beyond control. The outcome of the analysis is a set of *key disadvantages* chosen amongst the revealed causes, considered to be the most appropriate to address in order to eliminate the target disadvantages.

The procedure is documented with a diagram composed of boxes with descriptions of disadvantages and arrows indicating the flow of causality. The linear chains of causes are usually connected on inputs (with common causes) or on outputs (with logical operators), indicating how the contributing causes trigger the effects. Similarity between the structure of a CECA diagram and a combinational logical circuit using AND / OR gates inspired the concept we have originally presented in [4] and expanded in [5, 6] later on. It relies on representing the structure of a CECA diagram with a set of logical functions operating on root causes or intermediate disadvantages, perceived as Boolean variables, evaluating to 1 / 0 for an active / inactive cause, respectively.

Such a model allows the analysts to apply logical minimization and other techniques used for developing combinational logical circuits to support the selection of the key disadvantages. A drawback of this approach is the lack of time reference, inherited from the original CECA model, where the only notion of time comes from the arrows going from causes to effects, thus determining the sequence. In addition, the logical model only reflects the structure of the input diagram, ignoring the contents of the boxes, i.e. the specific disadvantages involved. To address these shortcomings, in [7] we have proposed modeling of CECA diagrams using Hierarchical Concurrent Finite State Machine paradigm [8] and the rules outlined in the next section.

2 Building state machine CECA model

The main assumption of the proposed state machine approach is that a CECA diagram describes interconnected *harmful processes*, i.e. sequences of operations performed within the system or super-system that jointly “produce” the target disadvantages. This seems to closely correspond with the concept of a *harmful machine* proposed in [9]. Each of these processes is modeled with a linear chain of causes and the connections between the chains (common causes and logical operators) indicate specific conditions, required for the “production” to progress. It is also assumed that the input CECA model is developed using the condition-action style devised in [10], i.e. it contains interleaved boxes representing interactions and conditions. A CECA model structured this way may be transformed into a state machine model, with nodes reflecting *states* and labelled edges reflecting conditional *transitions*, using the following guidelines [7]:

- nodes representing *actions* in the input diagram are converted into respective *states* in the state machine diagram,
- nodes representing *conditions* in the input diagram are converted into *transitions* with respective condition labels, positioned accordingly in locations of the incoming and outgoing edges of the original diagram,
- *common causes* being condition nodes, connected to several succeeding nodes in the input diagram, are reflected in the state machine diagram as *groups of edges* modeling transitions to respective states (labelled with the same condition inherited from the original cause),
- *OR operators* appearing in the input diagram are converted into *groups of edges* in the state machine diagram (one edge for each input), modeling *alternative* conditions required for transitions to the respective output states; in practice OR operators are often omitted and depicted as multiple edges, which do not need conversion,
- *AND operators* appearing in the input diagram are converted into *additional nodes and edges* in the state machine diagram, modeling *coincidence* of conditions required for transitions to the respective output states,
- a *loopback edge* is created for each of the nodes in the state machine diagram, with a condition complementary to conditions of all other outgoing edges of this node to model waiting in the same state; such a transition is default when none of the exit conditions are met and it is usually not shown in diagrams.

These instructions are illustrated in Fig. 1, together with two additional rules:

- *feedback paths* (“vicious circles”) in the input diagram are converted into respective cyclic transitions in the state machine diagram, as for regular action-condition segments,
- *common causes* being action nodes, connected to several succeeding nodes in the input diagram, are reflected in the state machine diagram as *groups of edges* modeling transitions to respective states (labelled with the conditions inherited from the corresponding target nodes).

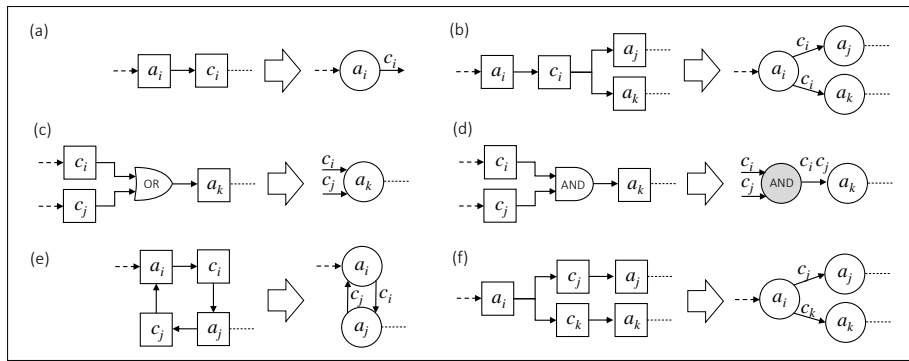


Fig. 1. Building blocks of a CECA diagram and their counterparts in a state machine model [7]: regular action-condition segment (a), action-condition segment with a common cause (b), OR operator (c) and AND operator (d). Additional building blocks: feedback connection (e) and branching to multiple next states with different conditions (f). Concatenation of symbols denotes logical AND. Loopback edges have been omitted for clarity.

3 Simplifying state machine CECA model

It was shown in [7] that above-mentioned conversion results in a decreased number of nodes in the diagram. This effect comes from changing the nodes reflecting conditions into edges representing conditional transitions. On the other hand, AND operators were also transformed into states, accordingly to the original method. It is worth noting, however, that an AND operator in a CECA diagram does not reflect any interaction and serve logical purposes solely, as it indicates that the output effect is triggered when all contributing causes are active.

Consequently, the state inherited by the state machine model from an AND operator does not reflect interaction either. It is only used to represent a stage where the process waits for a specific combination of conditions before progressing to the next stage. In terms of the synchronization scheme, it may be considered as deferring the transition until the last input cause becomes active. Such approach to AND conversion violates the clear differentiation among actions (represented by states) and logical conditions (represented by transitions), thus introducing a deficiency of the original method.

A solution to this problem is depicted in Fig. 2. An AND-state may be removed from the model if the transitions from all the input states are reconnected to the output state with conditions equal to logical AND (conjunction) of the original input conditions. In the example the AND-state inputs are c_i coming from a_i and c_j coming from a_j while the output transition to a_k requires $c_i c_j$ condition. After simplification both transitions from a_i and a_j to a_k require $c_i c_j$ condition and AND-state disappears.

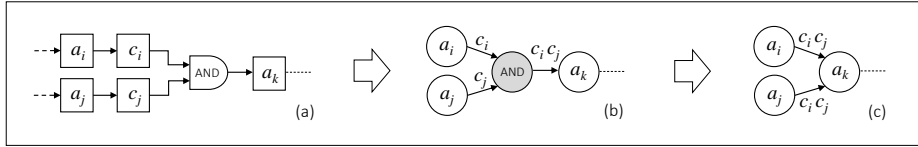


Fig. 2. Simplifying AND operator representation: original fragment of a CECA diagram (a), state machine model representation proposed in [7] (b) and minimized representation without artificial AND-state (c).

The basic rule for minimizing state machines indicates that two states may be unified (merged) if and only if their output functions are identical and their transition functions are compatible (identical or at least non-contradicting). Simply put, this requires that observed output behavior, as well as pattern of conditional transitions to the next states, are the same before and after the minimization. In the area of digital design this rule is used to decrease the number of states in order to simplify circuit construction and state encoding. Let us analyze if and how does this concept correspond to a state machine CECA model.

Identical output functions in an abstract state machine map onto identical actions in a sequential CECA model, i.e. pairs of interacting objects (tools and products) and the operations must be identical for both the candidate states. As for the output transitions, there are several generic variants possible. Identical conditions are conditions referring to same parameters, relations and threshold values, which is a special case of equivalent conditions, evaluating to equal logical values in all situations – even if they are formulated differently in the input model. This is illustrated with fully overlapped circles in Fig. 3c. Contradicting and non-overlapping conditions (Fig. 3a) never evaluate to true at the same time – e.g. $T < 25$ and $T > 30$. Partially overlapping conditions (Fig. 3b) only for some cases both evaluate to true – e.g. $T > 25$ and $T < 30$. For nesting (Fig. 3d) one condition is “stronger” than the other – e.g. $T < 25$ and $T < 30$, while complementary conditions (Fig. 3e) do not overlap and cover all situations – e.g. $T < 25$ and $T \geq 25$.

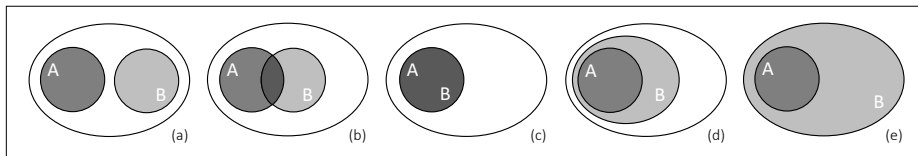


Fig. 3. Different combinations of conditions: contradicting / non-complementary (a), partially overlapping (b), equivalent / fully overlapping (c), nested (d), contradicting / complementary (e).

The simplest case of state merging is when the next states of both the candidates are the same. This situation is schematically depicted in Fig. 4.a. In such a case the states a_i and a_j are merged into a single state, while the transitions c_i and c_j are merged into a single transition with condition c_i+c_j describing an alternative (logical sum) of the conditions. If each of the merge candidates has a different next state, the states are merged like in the first variant described above and the conditions guarding particular transitions are preserved (Fig. 4.b). As can be seen, the first scenario leads to combined conditions, while the second scenario leads to combined sets of the next states.

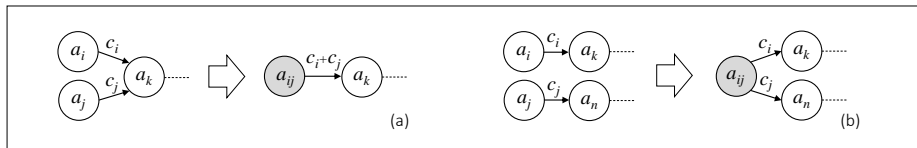


Fig. 4. Generic variants of state merging: for same next state (a), for different next states (b). Single next states are shown for simplicity; for multiple next states the rules apply respectively.

Depending on the relation between the conditions involved, these two scenarios may generate several specific cases, summarized in Table 1. As stated in [7], each linear chain of an input CECA model is represented by a separate state machine with an initial state reflecting respective root cause and all such state machines operate concurrently. An important consequence of this approach is the possibility of simultaneous triggering of several next states through separate transitions labelled with the same condition. This extended interpretation implies, that any two states representing same interaction may always be merged, because their transition functions are always compatible.

Table 1. Possible configurations of conditions and next states of the merge candidates.

conditions	identical next states	different next states
(a) non-overlapping non-complementary	conditional transition with sum of conditions	conditional branch mutually exclusive (none, one or another next state)
(b) partially overlapping non-complementary	conditional transition with sum of conditions	conditional branch exclusive or concurrent (none, one, another or both states)
(c) fully overlapping non-complementary	conditional transition with either of conditions	conditional branch always concurrent (none or both next states)
(d) one nested in another non-complementary	conditional transition with “weaker” condition	conditional branch exclusive or concurrent (none, one or both next states)
(e) non-overlapping complementary	unconditional transition (sum of conditions always evaluates to true)	immediate conditional branch mutually exclusive, obligatory (one or another next state)

To conclude this section, it should be noted that a state machine model obtained from a converted CECA diagram may be simplified by removing the artificial AND-states as well as by merging the states reflecting the same interactions in the analyzed system.

4 State machine model vs. formal language

The concept of a formal language [11] comes from the field of Mathematical Linguistics and refers to a language generated (defined) by a formal grammar. There are several categories of formal languages, with different limitations regarding the generation rules. The most restricted are *regular languages*, generated by *regular grammars*, which are recognizable by Finite State Machines (FSM). A formal grammar is defined by indicating:

- set of *terminal symbols* (terminals),
- set of *non-terminal symbols* (non-terminals), containing a designated *start symbol*,
- set of *productions*, transforming sequences of symbols into sequences of symbols.

Non-terminals are symbols to be replaced with sequences of symbols determined by productions. The generation of an expression in a target language begins with a start symbol and uses particular productions to convert non-terminals in the successive output sequences, until the sequence contains terminal symbols solely.

An example of a simple grammar is given below. Using productions 1 and 3 we obtain expression “zm” and using the productions 1, 2, 2, 3 we obtain “zoom”. As can be seen, this grammar defines a language containing all expressions starting with the “z” symbol, ending with the “m” symbol and containing zero or more “o” symbols in between – i.e. “zm”, “zom”, “zoom”, etc.

```

terminals:      {z, o, m}
non-terminals: {S, T}, start symbol S
productions:   1. S → zT    2. T → oT    3. T → m

```

An alternative way of defining a set of expressions sharing common morphological characteristics uses notation called regular expressions [11]. Such expression describes a template of all target expressions and it may contain some terminal symbols and some special symbols indicating production rules implicitly. For instance, an asterisk denotes that preceding symbol may appear zero or more times, so that sample grammar defined above is represented by the regular expression: “zo*m”.

As stated before, a state machine may be used to recognize particular grammar, i.e. to check if a sequence of symbols presented at the input constitutes a valid expression in this grammar. This requires that some states are designated “accepting states” and whenever any of such states is reached, the input sequence is considered valid. In other words, for each regular grammar an equivalent finite state machine may be created.

What we intend here, with regard to cause-effect analysis, is the opposite: we want to find a grammar or regular expression representing a given state machine CECA model. The target disadvantages are logical counterparts of the accept states in this case. And we are looking for a synthetic linguistic description of all paths existing in the model that lead from the root causes to the target disadvantages. Expected outcome of this approach is a compact specification (or a “prescription”), stating what exactly and in what order must happen in the analyzed system to generate all identified target disadvantages.

A definition of a state machine recognizing the sample grammar is presented below. It is described using the notation required by an online FSM simulator [12], with simplified formatting (originally, each entry should be put on a separate line). The diagram automatically generated by the application from this definition is shown in Fig. 5a.

```
#states      0  1  2  3
#initial     0
#accepting   3
#alphabet    z  o  m
#transitions 0:z>1  1:o>2  2:o>2  1:m>3  2:m>3
```

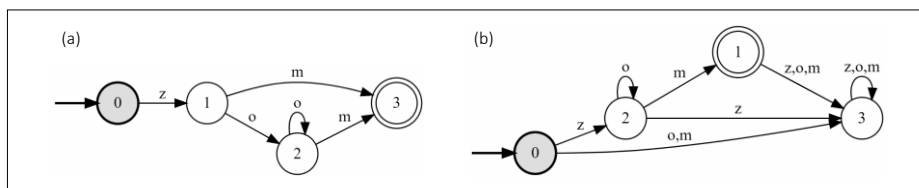


Fig. 5. State machine diagrams generated by application [12]: from the explicit definition of the state machine (a) and from the regular expression “zo*m”, as a grammar recognizer (b).

As can be seen, the accept state 3 may be reached from the initial state 0 through state 1 or through states 1 and 2, reflecting various production rules defined by the grammar. Such an automaton only describes the transitions expected for valid expressions and the diagram of a complete recognizer generated by the application for the regular expression “zo*m” is shown for reference in Fig. 5b. It introduces two important changes:

- an additional state is used for signaling invalid expressions violating the template,
- valid expressions are detected using smaller number of states than in grammar-based implementation (3 instead of 4), as the state 2 recognizes 0 or more “o” symbols.

5 Discussion

Let us begin the discussion with summarizing the main properties of the two modelling approaches. The classic (structural) CECA model features:

- three types of diagram components – boxes, logical operators and arrows,
- boxes represent disadvantages – whatever their nature is,
- logical operators indicate how do the causes combine to trigger the effects,
- arrows represent causality flow and do not have any attributes.

State machine (behavioral) CECA model features:

- two types of diagram components – boxes and arrows,
- boxes represent states reflecting the interactions within the system or super-system,
- arrows represent conditional transitions between the states, indicating causality flow, and they are labelled with the logical conditions.

Next, let us focus on the differences between the state machine approach and the classic CECA method using four perspectives of the SWOT analysis.

Strengths:

- states and conditional transitions represent behavior of the concurrent processes better than a structural diagram, which looks static in comparison,
- stages of the process and transitions between stages are clearly distinguished with dedicated types of diagram elements, which makes the model more comprehensible,
- logical operators are converted into states or transitions and disappear as a separate type of nodes, which makes the model simpler,
- state machine representation is more compact and therefore more expressive than a structural diagram with unlabeled arrows,
- state machine approach is much more disciplined than classic CECA, which uses guidelines rather than strict rules, leaving a lot of space for experience and style,
- condition-action duality is supported and enforced at the level of the state machine notation, while for structural diagrams it is only a modeling convention,
- state machine model may be minimized in a systematic way, with the states and logical conditions processed algorithmically,
- model correctness may be verified in a more orderly way than for structural model, by checking logical coherence of the conditions,
- transitions labelled with same conditions indicate synchronization points between concurrent processes reflected by particular state machines in the model,
- synthetic description of the harmful processes may be extracted from a state machine model in the form of a regular expression.

Weaknesses:

- state machine approach is more complicated and constrained than regular CECA, where anything considered a disadvantage may be included in the model,
- proposed theoretical model uses several unintuitive concepts, like many states being current and active at the same time [7] – structural CECA modeling is easier,
- the procedure for creating state machine CECA model from scratch has not been devised yet, so that for now the structural model has to be built first.

Opportunities:

- explicit references to transitions allows the analysts to focus on the configurations of conditions in time, possibly inspiring new solutions – e.g. desynchronizing events,
- state machine approach is well known in the IT and other engineering areas, which increases chances of successful communications with specialists in these areas,
- existing state machine notations and tools facilitate automatic processing of model descriptions (e.g. extraction of the regular expressions).

Threats:

- structural CECA method have been used and taught within TRIZ community for decades, hence the current demand for a new approach seems to be relatively low.

6 Example

We will use the sample CECA diagram discussed previously in [7]. As shown in Fig. 6, the original graph with 24 nodes and 24 edges is firstly converted into a state machine with 14 states and 14 transitions. Then the three AND-states are removed, as described in section 3, yielding 11 states and 11 transitions. Finally, the conditions are mapped onto single symbols for obtaining state machine description in the linguistic form: $c_1 \rightarrow a$; $c_2 \rightarrow b$; $c_3 \rightarrow c$; $c_5 \rightarrow d$; $c_7 \rightarrow e$; $c_1c_6 \rightarrow p$; $c_3c_4 \rightarrow q$; $c_7c_8 \rightarrow r$.

The regular expression representing all scenarios leading to a given target disadvantage may be found by indicating the state reflecting this disadvantage as the accept state and checking the paths from all the initial states (reflecting the root causes). The regular expression representing all the harmful processes, obtained by aggregating the expressions found for all target disadvantages, reads as: “ $p + bp + cp + a + qe + qr + dr$ ”. This representation indicates 7 different scenarios generating the target disadvantages after encountering 1 or 2 specific combinations of conditions within each scenario.

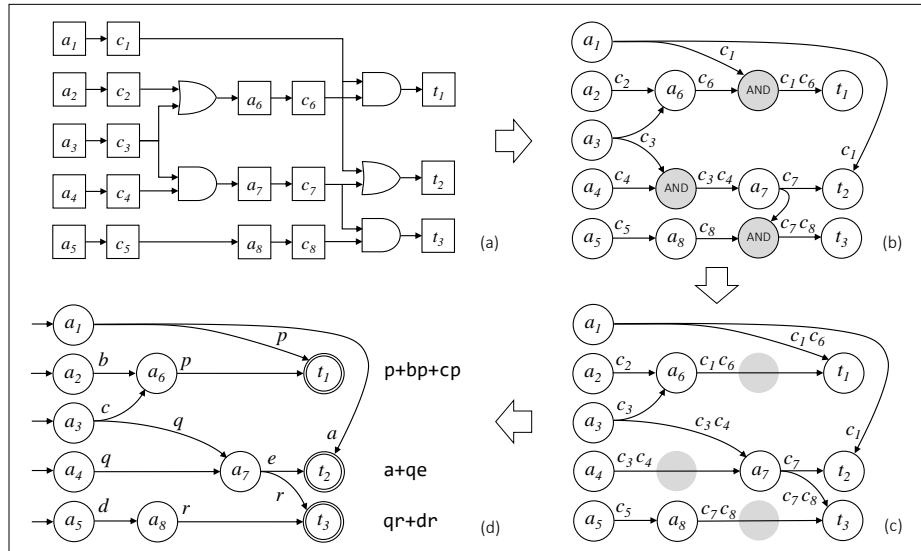


Fig. 6. Sample CECA diagram (a) and equivalent state machine diagram (b) as described in [7]. The same diagram with removed AND-states introduced during conversion (c) and transformed into form usable for linguistic approach (d). States representing target disadvantages are annotated with respective regular expressions.

7 Summary and further work

We have recalled the previous research on state machine CECA modeling and extended the results by indicating additional diagram conversion rules, as well as introducing and discussing two new topics: state model minimization and its linguistic representation. We have also compared the state machine model with the classic CECA diagram.

As can be seen in the example and the preceding sections, the linguistic approach focuses on the transitions and seems to neglect or ignore the intermediate disadvantages inherited from the original CECA diagram in the form of states reflecting interactions. This is an interesting intensification of distinguishing interactions and conditions in the model. To analyze it in a systematic manner, we will start with a short summary:

- the states in the state machine CECA model reflect actions and transitions reflect conditions required to progress the “harmful processes”,
- the initial states reflect root causes (they are always active, such as gravity or law) and terminal (accept) states reflect “final products”, i.e. target disadvantages,
- the states model how the things are going or may be going – i.e. existing or future harmful interactions between the components of the system or its super-system,
- the transitions indicate which operations, in what order and upon what conditions will produce particular target disadvantages.

Using this allegory of a production process, we may continue with the following:

- all the intermediate states jointly describe the “production means” or capabilities of the “factory” – which seems to fit in well with the harmful machine concept,
- the sequences of transitions form “prescriptions” or “instructions” for using these capabilities to develop respective target disadvantages,
- distinguishing capabilities (states) and instructions (transitions) looks very much like the hardware-software duality of the computer systems,
- regular expressions extracted from a state machine CECA model provide compact and complete representation of these instructions.

The success of the computer technology comes to a great extent from the ability to change the operation of the programmable hardware by changing the software solely. And therefore it presumably makes sense if we first focus on the “harmful program” controlling the behavior of the harmful machine, as it might be easier to introduce changes in this layer, rather than changing the machine itself.

The traditional approach to system improvement employs elimination of the key disadvantages. The enhanced perception of the logical conditions and time relations seems to open a wider perspective. Perhaps in addition to *removing* something from the model it is possible to get rid of the target disadvantages by *changing* some transitions as well? Maybe we could desynchronize some interactions in particular harmful processes or change their order of appearance or implement self-blocking or cross-blocking between the processes?

These questions indicate that the area for future research regarding the state machine approach and condition-action duality in cause-effect analysis is extensive. It would also be interesting to coordinate this work with other CECA-related activities [13, 14].

Acknowledgments. Author gratefully acknowledges Dr. Oleg Abramov for inspiring discussions regarding the CECA method and its extensions. Mr. Dariusz Burzyński should be credited for his linguistic support.

References

1. Litvin, S.S., Akselrod, B.M.: Cause-Effects Chains of Undesired Effects, Methodical Theses (in Russian). CPB, 1996.
2. Abramov, O., Kislov, A.: Cause-Effect Analysis of Engineering System's Disadvantages, Handbook on Methodology (in Russian), Algorithm, Ltd., 2000.
3. Abramov O.: TRIZ-Based Cause and Effect Chains Analysis vs Root Cause Analysis. In: Souchkov, V., Kässi, T. (eds.) Proceedings of the TRIZfest-2015 International Conference. Seoul, South Korea, pp. 288-295. MATRIZ (2015).
4. Chrząszcz, J., Salata, P.: Cause-Effect Chains Analysis using Boolean algebra. 16th International TRIZ Future Conference TFC 2016. In: Koziółek S., Chechurin L., Collan M. (eds.) Advances and Impacts of the Theory of Inventive Problem Solving. The TRIZ Methodology, Tools and Case Studies, 2018, ISBN 978-3-319-96531-4, pp. 121-134.
5. Chrząszcz, J.: Quantitative approach to Cause-Effect Chains Analysis. In: Souchkov, V. (ed.) Proceedings of the TRIZfest-2017 International Conference. Krakow, Poland, pp. 341-352. MATRIZ (2017).
6. Chrząszcz, J.: Indicating system vulnerabilities within CECA model. In: Mayer O. (ed.) Proceedings of the TRIZfest-2018 International Conference, Lisbon, Portugal, pp. 31-37, MATRIZ 2018.
7. Chrząszcz, J.: Modelling CECA diagram as a state machine. In: Cavallucci D., DeGuio R., Koziółek S. (eds.) Automated Invention for Smart Industries, 18th International TRIZ Future Conference TFC 2018 Proceedings, IFIP AICT, vol. 541, 2018, ISBN 978-3-030-02455-0, pp. 302-314.
8. Girault, A., Lee, B., Lee, E.A.: Hierarchical Finite State Machines with Multiple Concurrency Models. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 6, June 1999, pp. 742-760.
9. Lenyashin, V., Kim, H.J.: "Harmful System" – using this concept in modern TRIZ (in Russian). 2006. <http://www.metodolog.ru/00859/00859.html>, last accessed 2019/06/30.
10. Yoon, H.: Occasion Axis and Parameter-Function Pair Nexus for Effective Building of Cause Effect Chains. In: Souchkov, V., Kässi, T. (eds.) Proceedings of the TRIZfest-2014 International Conference, Prague, Czech Republic, pp. 184-194. MATRIZ (2014).
11. Kandar, S.: Introduction to Automata Theory, Formal Languages and Computation. Pearson India, 2013; ISBN: 9788131793510.
12. Zuzak, I.: FSM simulator. http://ivanzuzak.info/noam/webapps/fsm_simulator/, last accessed 2019/06/30.
13. Chrząszcz, J.: Deriving quantitative characteristics from CECA model. To be presented during TRIZfest-2019 International Conference, Heilbronn, Germany (2019).
14. Chrząszcz, J.: Logical negation in CECA model. To be presented during TRIZfest-2019 International Conference, Heilbronn, Germany (2019).