Modelling CECA Diagram as a State Machine

Jerzy Chrząszcz

Modelling CECA diagram as a state machine

Jerzy Chrząszcz[1,2]

[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. Cause-Effect Chains Analysis (CECA) is one of the main TRIZ methods used for identification of system disadvantages. The analysis results in a diagram documenting these disadvantages and causal relations between them. Although the nature of causality implies that any effect must follow its cause, the original CECA concept does not address time explicitly. This drawback has been indicated by Yoon, who proposed *Occasion Axis* to describe changes of system state upon meeting particular conditions specified using values of parameters. Such an axis illustrates a sequence in time and an additional requirement is to interleave nodes referring to parameters with those referring to functions, originally dubbed as *Parameter-Function Pair Nexus*.

The method of transforming a CECA diagram into a logical model presented during TFC 2016 conference relies on decomposing the diagram into a context-dependent layer (specific content) and a context-independent layer, representing the structure of connections. The logical model describes the structure with a set of Boolean functions, which may be minimized and analyzed in a systematic way.

This paper explores the idea of Occasion Axis and examines the possibility of converting a CECA model into a state machine with transitions between the states described by conditions referring to parameters of objects in the analyzed system or its super-system. The expected benefits of such transformation range from better understanding of the time-domain interrelations of the causes up to describing the causality using standard notation, such as UML. The paper presents rules proposed for systematic conversion of CECA diagram into state machine representation and discusses required extensions to formal state machine definition.

Keywords: Cause-Effect Chains Analysis, logical model, Boolean algebra, state machine, harmful process.

1 Basic CECA model

Cause-Effect Chains Analysis (CECA) is an iterative method for revealing causal relations in the analyzed system [1, 2]. It starts with indicating drawbacks to be removed, which are called *target disadvantages*. Then their causes (*intermediate disadvantages*) are investigated subsequently, until finding primary causes (*root causes*) that reflect laws of nature or specific constraints of the project, remaining

beyond control. Because root causes cannot be literally eliminated, CECA procedure aims at identification of the *key disadvantages* instead, removal of which should remove target disadvantages.

The analysis results in a diagram with boxes containing descriptions of disadvantages, arrows denoting causality flow and – possibly – logical AND / OR operators, reflecting how the causes contribute to a given effect. Although the nature of causality seems simple at the level of intuition and common sense, it is not equally straightforward for systematic modelling and analyzing, as may be seen in [3, 4] and the accompanying discussions.

Several doubts regarding the original method have been indicated in [5], addressing possible approaches to in-depth and in-width growth of causality diagrams, as well as identifying proper stop conditions, which jointly affect completeness of the diagram. Significant improvements to the method employ using cause-effect patterns identified in real projects [6, 7], adding information about advantages (positive effects) in addition to disadvantages [8], observing specific structure of the causal chains to support their correctness and completeness [2, 9] or using additional criteria [8, 10].

2 Logical CECA model

Systematic approach to conversion of CECA diagram into a set of logical functions is presented in [11]. It starts with decomposition of the diagram into two separate layers: contents and structure. The contents (box descriptions) are specific to a particular problem situation, while structure of the interconnections is context-independent. Proposed logical model uses binary logic with 0 representing inactive disadvantage and 1 representing active disadvantage. Logical operators are modelled with respective gates. AND gate outputs 1 only for 1s on all its inputs, while OR gate outputs 0 only for 0s on all inputs, and the truth tables of 2input gates are as follows:

AND

x	y	$x\,y$
0	*0*	*0*
0	*1*	*0*
1	*0*	*0*
1	*1*	*1*

OR

x	y	$x + y$
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*1*

Target disadvantages are outputs of the network and the inputs may reflect root causes or any intermediate causes – in particular, the key disadvantages. Such approach allows for describing structure of the CECA diagram with a set of combinatorial logical functions – one for each of the target disadvantages. These functions may be transformed using rules of Boolean algebra and minimized in order to obtain most concise representation of the logical relations between selected input causes and target disadvantages. This paves the way to answering important questions regarding the structure of the model of causality, ranging from *which inputs influence*

particular output? up to *what minimal set of inputs must be acted upon in order to deactivate all the outputs?*

Some enhancements to the logical model were introduced in [12], including provisions for removal of intermediate disadvantages and criteria for verification of model completeness. Proposed representation was developed as a solution to a physical contradiction describing disadvantages in linear CECA chains (excluding the root causes):

- each disadvantage should have *exactly one control* to retain coherence between the model and the diagram (as we only draw one arrow between the boxes), BUT
- each disadvantage should have *more than one control* to retain coherence between the model and the concept (as we may remove a disadvantage without removing its predecessor).

Using *separation in relation* principle [13] the active / inactive status of the intermediate disadvantage was split between the input (independent of deeper causes) and the output (dependent on deeper causes). The logical model of a linear chain of disadvantages comprises of cascaded AND gates, with one input connected to the output of previous gate and the other input controlled by an independent variable, as shown in Fig. 1. The first input is used for passing information about deeper causes in the chain and the second input controls given disadvantage directly. If any of the inputs is disabled (i.e. set to 0), the output is disabled as well, simulating *indirect* and *direct* removal of the given disadvantage, respectively.

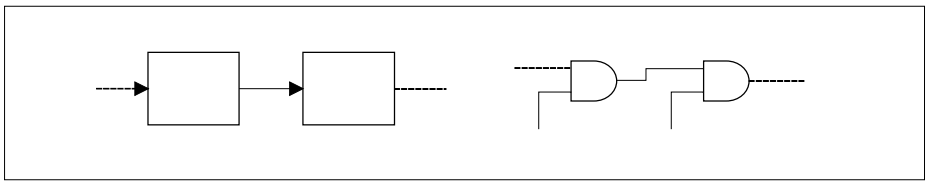

Fig. 1. Chain of AND gates modelling linear branch of a CECA diagram [12].

3 Time axis and Occasion Axis

Although the nature of causality implies that any effect must follow its cause, the original CECA concept does not address time explicitly and the only notion of time precedence comes from arrows pointing from causes to effects. Such representation clearly defines a sequence of disadvantages in a given linear chain, but it tells nothing about the time relations between disadvantages located in different branches of a diagram.

This also holds true for the logical model considered so far, consisting of combinatorial functions i.e. with outputs depending solely on inputs. When a change is applied to an input of a combinatorial network, it propagates through subsequent gates until it reaches a particular output or disappears on the way due to a specific logical function. For an AND gate 0 on any input switches output to 0 independently

of other inputs, so that AND output will reflect changes of a particular input if and only if all remaining inputs are 1s. Similarly, for an OR gate 1 on any input switches output to 1, hence the change of a particular input will propagate to OR output if and only if all other inputs are 0s. In the digital design area this approach is called *path sensitization* and it is used in testing to assure that changes of a given signal will propagate to an observable output.

The time axis is explicitly used in [14] for modelling causal chain as a sequence of events, which is similar to extending a single row of screens in the System Operator into the past (to identify the causes) and into the future (towards a solution). An obvious limitation of this method is the inability to model multiple connected chains of causes.

The lack of a strict notion of time in CECA method was extensively explored by Yoon [9], who pointed out that disadvantages occur due to changes of particular parameters of a system (or super-system), not just because of the time flow. The proposed approach uses concept of *occasion*, being a moment in time when a particular parameter pertinent to the analyzed problem situation has a certain value. And the *Occasion Axis* is a sequence of occasions in time that describes the development of a target disadvantage.

Points on this axis are determined by the nature and intensity of the interactions e.g. the moment of thermal shutdown of a computer is determined by a preset temperature limit and operating conditions, and so its location in time may vary. Depending on the amount of generated heat and the heat dissipation efficiency, the shutdown may be performed sooner or later, but it follows one scenario. And lack of the shutdown during a long operation also fits within the same scenario, yet unfinished. Therefore Occasion Axis may be considered discrete time axis defined by moments when some threshold values of pertinent parameters are achieved (i.e. specific events happen).

Furthermore, Yoon argues that a properly modelled chain of causes should consist of interleaved nodes of two types, defining conditions and actions. Conditions should refer to particular values of parameters, while actions should describe interactions between objects. The resulting chain was dubbed *Parameter-Function Pair Nexus*.

The justification for this interleaved structure is that a state of an object may only be changed as a result of interactions between the objects. Therefore neither two states nor two actions may appear in the diagram one after another and subsequent nodes of the same type indicate a missing node of the opposite type, which should be inserted in between. Recommended forms of descriptions are as follows [9]:

- state (condition): *entity* + *its parameter* + *value of the parameter*,
  e.g.: temperature of processor is higher than *Ts*,
- action (function): *tool* + *action* + *object*,
  e.g.: processor heats computer case (excessively).

Such structured approach to construction of the cause-effect chains allows for detection of other types of omissions, including overlooked parameter-function or function-parameter pairs. This results in additional support for ensuring completeness

of the CECA diagram. Nevertheless, all information regarding the causes still comes from the contents of boxes and the arrows only indicate the flow of causality.

Time is also referred to in Root Conflict Analysis (RCA+) developed by Souchkov. The method is focused on both disadvantages and advantages, possibly brought by some of the causes identified during analysis [8]. This allows for indicating physical contradictions with conflicting requirements to be fulfilled "at the same time". Time references are used at the stage of selecting root conflicts, but they reflect conditions rather than specific moments in time (e.g. "during strong wind"), which seems similar to the concept of occasion. Another similarity between RCA+ and Yoon's approach comes from the recommendations for describing identified causes, which also distinguish conditions and functions.

4 Merging Occasion Axis with logical model

The question usually asked when building CECA diagram is *why* which seems adequate and correct in this context. But adding another box and arrow to a cause-effect chain is like answering only one part of this question, namely *what* causes a particular effect, while we are also interested in *when* and *how* does this "causing" happen.

A renowned approach capable of modelling conditional changes of a system in time is *state machine* representation. Therefore it seems interesting to attempt a conversion of a CECA diagram into a state machine (automaton) with finite number of states and deterministic transitions between the states, i.e. *Deterministic Finite State Machine* (DFSM). State machines and other abstract machines are covered by Automata theory, and an introduction to this topic pertinent to the scope of our paper is given in [15].

DFSM is formally defined as a 5-tuple $< Q, \Sigma, \delta, q_0, F >$, where:

- Q is a finite set of states,
- Σ is a finite set of input symbols,
- δ is the transition function ($\delta: Q \times \Sigma \rightarrow Q$),
- q_0 is the initial state of the automaton ($q_0 \in Q$),
- F is a set of states called accept states ($F \subseteq Q$).

Informally, we need some *states* to be defined as well as conditional *transitions* between the states, which depend on inputs. Combinations of inputs form *input symbols* and the *initial state* is the predefined start state, in which the automaton awaits the first input symbol. Outputs of an automaton depend on its state and – in particular – may differentiate the *accept states* and the other states. Automata are usually depicted as graphs with nodes representing states and directed edges (arcs) representing transitions.

The simplest state machine implementation used in digital circuits is the D-type flip-flop with two states {zero, one} and two input symbols {0, 1}. Transition function makes the flip-flop state to follow the input, i.e. 0 on input causes transition

to state zero with output 0 and 1 on input causes transition to state one with output 1. In other words, such flip-flop remembers the latest input symbol and so it may be considered a 1-bit memory element. To complete the definition, we select zero as the initial state and one as the accept state.

Respective transition graph is shown in Fig. 2 using generic notation and basic UML notation for state machine modelling. The "E" prefixes in the UML model denote that respective actions are performed upon entering given states (as this notation also allows for describing actions executed upon exiting the states).

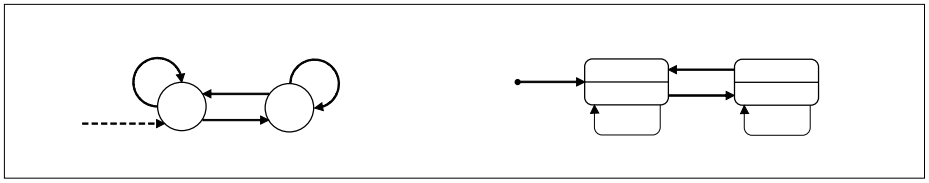

Fig. 2. State diagram (left) and basic UML model (right) of D-type flip-flop.

An example of an automaton with 4 states, 2 binary inputs and 1 output is shown in Fig. 3. The set of input symbols contains 4 elements {00, 01, 10, 11} and edge labels describe input symbols. The required operation is to detect the sequence 11-00-11.
The initial state is denoted as A and the accept state is denoted as D. The successful detection is indicated with 1 on the output and all other states outputs 0. Response of the state machine to a test sequence 00-01-11-00-11-00-11-10 is presented below.

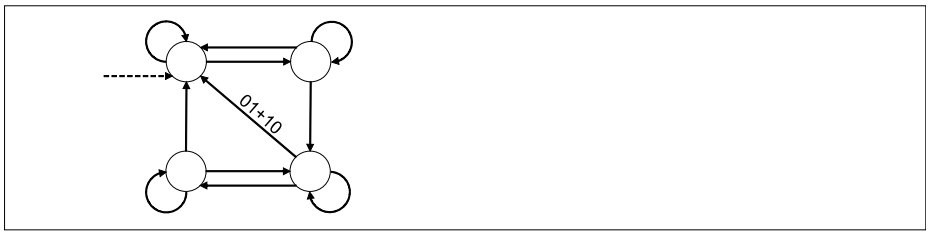

Fig. 3. Sample state machine and description of its operation. Plus sign denotes logical OR.

In taxonomy of the logical circuits state machines are categorized as sequential circuits. Outputs of a sequential circuit depend both on its inputs and its current state [15]. That is why a state machine (contrary to combinatorial logic) may produce different responses to the same input signals, and in the above example the first 11 symbol in the sequence (t2) could be distinguished from the second (t4). It is worth noticing that the third occurrence (t6) also matches the pattern, so that state D is traversed twice.

5 Converting CECA diagram into state machine diagram

An apparent similarity between the state machine diagram and the CECA diagram is both promising and misleading. The former suggests an ability to reflect a structure of causal interrelations between disadvantages as states and transitions, while the latter comes from significant semantic differences between the respective elements:

- nodes in a CECA graph describe disadvantages, while nodes in a state machine graph describe states of the automaton,
- edges in a CECA graph only indicate direction of the causality flow (and so they do not need any labels), while edges in a state machine diagram describe transitions (and they are labeled with respective input symbols or logical conditions),
- logical operators in a CECA diagram reflect how the input causes combine to trigger the resulting disadvantages, with no direct counterpart in the state machine diagram.

Regardless of these differences, it should be noticed that:

- linear chain of disadvantages in a CECA diagram may be perceived as a model of the process of development of the last disadvantage in that chain,
- logical OR indicates that any of the input causes is sufficient to trigger the effect,
- logical AND indicates that all of the input causes are necessary to trigger the effect,
- state machine approach may be used for modeling processes.

From such perspective, we may interpret a CECA diagram as a model of the development of all target disadvantages included in that diagram. And this resembles the idea of a *harmful system* described in [16], being a conceptual source of the harmful functions resulting in the observed disadvantages of the analyzed system. By analogy, CECA diagram may be interpreted as a model of interconnected *harmful processes*, which – in spite of being unintended – "produce" unwanted effects in an organized and repeatable way.

This approach allows for transforming a CECA diagram into a state machine model, with the only prerequisite of having the diagram build using parameter-function (or condition-action) paradigm introduced in [9]. The proposed rules of conversion are described below and illustrated in Fig. 4.

- nodes describing *actions* in the CECA diagram are converted into respective *nodes* in the state machine diagram,
- nodes describing *conditions* in the CECA diagram are converted into *edges* with respective condition labels, positioned accordingly to locations of the incoming and outgoing edges in the CECA diagram,
- common causes, i.e. causes forking through edges to several nodes in the CECA diagram, are reflected in the state machine diagram as *groups of edges* modelling transitions to respective states (labelled with the same condition),
- OR operators appearing in the CECA diagram are converted into *groups of edges* in the state machine diagram (one edge for each input), modelling *alternative* of

conditions required for transitions to the respective output states; in practice ORs are often omitted and modelled as multiple edges, which do not need conversion,
- AND operators appearing in the CECA diagram are converted into *additional nodes and edges* in the state machine diagram, modelling *coincidence* of conditions required for transitions to the respective output states,
- an additional *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 transition is default when none of the exit condition are met and it is usually not shown in diagrams.

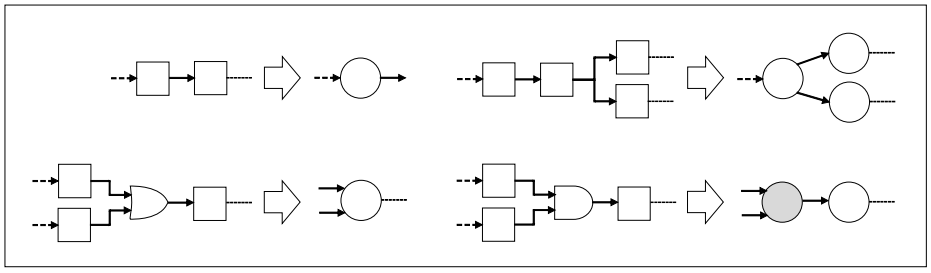

Fig. 4. Building blocks of a CECA diagram and their counterparts in state machine model: regular action-condition segment (a), action-condition segment with common cause (b), OR operator (c) and AND operator (d). Concatenation of symbols denotes logical AND. Loopback edges have been omitted for clarity.

A sample CECA diagram and the result of its conversion into state machine diagram is presented in Fig. 5, with root causes $a_1 \div a_5$ and target disadvantages $t_1 \div t_3$. As may be seen, the number of states is reduced compared to the initial number of nodes in the CECA diagram (14 vs. 24), because the nodes representing conditions are transformed into conditional transitions between the states.

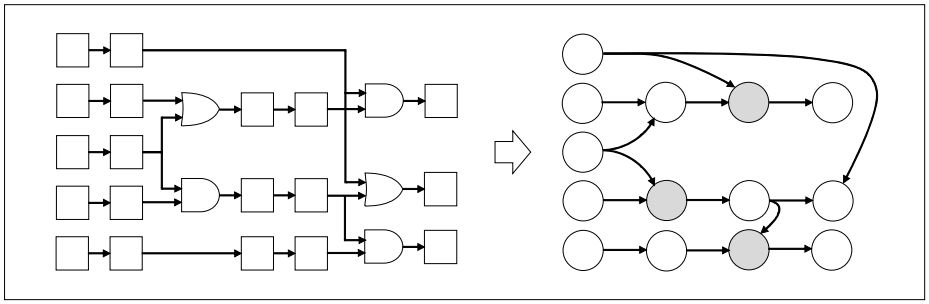

Fig. 5. A sample CECA diagram and an equivalent state machine diagram created using proposed conversion rules. Loopback edges have been omitted for clarity.

6 Analyzing sequential CECA model

Let us discuss the results of the conversion and the implications of the proposed rules.

Linear chain of causes. Described procedure properly transforms a linear chain of causes into a DFSM-like graph. The differences come from the specific interpretation of the states. For a regular automaton exactly one state is *current* and *active* at any given moment. In the transformed CECA model the states are related to functions performed in particular stages of a harmful process which "produces" a given disadvantage. Therefore the *current* state reflects the most advanced stage of the process reached within a chain (after a specific sequence of transitions), while some other states in this chain may be *active* i.e. their functions may still be performed.

Accept state of the automaton models the final product of the respective harmful process and therefore one accept (final) state should be generated during the conversion for each of the target disadvantages included in the CECA diagram.

Initial state of the automaton reflects the initial stage of the respective harmful process, which conforms to the criteria of identifying CECA root causes. Indeed, laws of nature or project constraints act continuously and thus qualify for the initial stages, when "production" of disadvantages has not started yet. And just like for the final states, we also need one initial state for each of the root causes included in the CECA diagram.

Input symbols are combinations of states of inputs and they are used for evaluating the transition function i.e. determining the next state of the automaton. Because transitions in the resulting state machine reflect conditions inherited from the CECA model, they should refer to particular objects, parameters and values. The inputs may be seen as logical signals evaluating to true or false – one for each of the conditions used in the state machine diagram, e.g. condition *temperature is higher than Ts* evaluates to true if $T > Ts$ and it evaluates to false otherwise.

Logical operators in the CECA diagram may be perceived as synchronization gates of the harmful processes. OR operator implies that the resulting transition will be triggered when the *first* of the involved conditions is satisfied, while AND operator implies that the transition will be triggered when the *last* of the involved conditions is satisfied. Hence an OR operator is modelled with separate edges in the state machine diagram (alternative transition for every OR input) and AND operator is modelled with extra state for waiting until all contributing causes become active. Some of these causes may be active and some may be inactive while automaton remains in the waiting state, which is coherent with the proposed interpretation of the current state.

Common causes are depicted in the CECA model with separate (or split) edges leading to two or more different resulting disadvantages. This is another type of a synchronization gate with one input and many outputs, deterministically triggered at the same time upon satisfying a particular condition.

There is a misunderstanding about referring to dependence between contributing causes for selecting logical operators in a CECA diagram. Some TRIZ materials recommend using OR operator "if underlying causes are independent of each other", while dependence (resulting, for instance, from a *common cause* of the input disadvantages) is not related to logical functions describing the influence on the output disadvantages. In other words dependence relates to causes and logical

operators relate to effects, so that both OR and AND may be used for dependent as well as independent causes.

Concurrency and hierarchy. Because a CECA diagram models a set of interconnected harmful processes, the resulting state machine is in fact a structured collection of automata running concurrently. The bottom level of the hierarchy is formed by linear chains, containing initial states, final states and – perhaps – some intermediate states, connected by respective conditional transitions. Such chains may contain root causes or target disadvantages and may connect on inputs (with common causes) or outputs (due to logical operators in the CECA diagram). And they may be treated as single *super-states* at the higher levels of the hierarchy. Taking the above into consideration, we should adjust the DFSM definition $< Q, \Sigma, \delta, q_0, F >$ presented before.

- set of states Q includes all stages of all processes modelled with linear chains in the CECA diagram and additional states resulting from conversion of AND operators,
- set of input symbols Σ is determined by all conditions inherited from the CECA diagram, so that all required transition criteria may be evaluated,
- transition function δ is determined by the locations and directions of the edges and OR operators in the CECA diagram; concurrent operation of automata is synchronized on transitions labelled with the same input symbol,
- instead of a single initial state $q_0 \in Q$ we need a set of initial states $Q_0 \subseteq Q$, which includes all root causes identified in the CECA diagram (each root cause determines the first stage of the respective harmful process),
- set of final states $F \subseteq Q$ includes all terminal stages of all linear chains in the CECA diagram – in particular, all target disadvantages.

The extended definition is close to *Hierarchical Concurrent Finite State Machine*, which may be systematically described using Unified Modeling Language, Place-Transition notation (Petri nets), state charts or other notations supported by modelling tools.

7 Example

As an example, we will consider computer overheating, which was briefly mentioned in section 3. Target disadvantage is that *computer stops because of overheating* and a simplified CECA diagram is shown in Fig. 6, together with a state machine diagram obtained using proposed conversion rules. The model covers two scenarios of creating excessive amount of heat and two variants of stopping the computer, depending on the operation of the thermal protection: properly configured (and operable) vs. disabled (or misconfigured, or inoperable). For clarity, the descriptions are shortened and the alternative causes in the CECA diagram are depicted with multiple edges instead of the explicit OR gates. As can be seen, interleaving of actions and conditions is crucial.

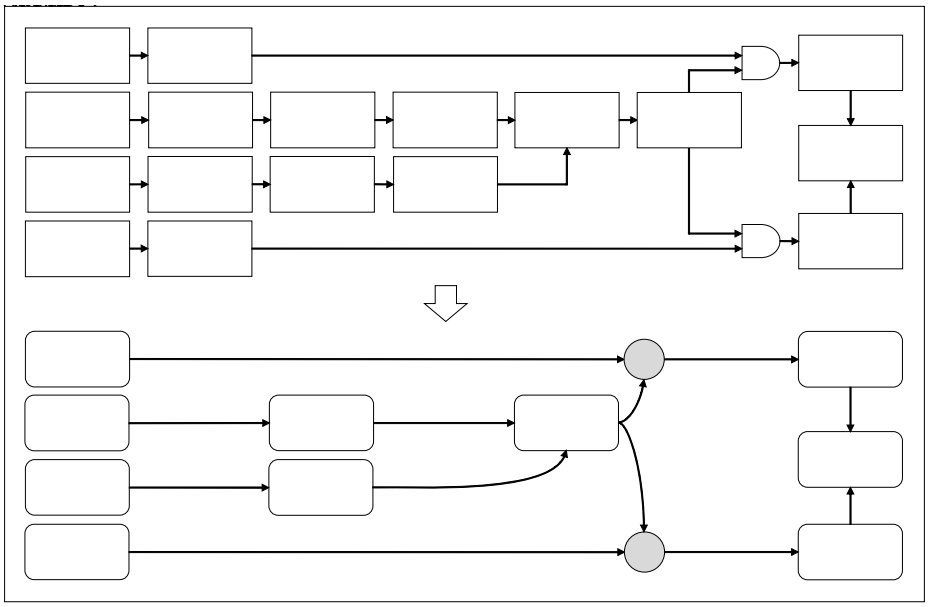

Fig. 6. An example CECA diagram (top) and the respective state machine diagram (bottom).

8 Summary and conclusions

We have briefly presented original CECA method and some improvements proposed in order to produce correct and complete models of causality. The Yoon's idea of the Occasion Axis and structuring the diagrams using condition-action segments provides proper perspective for modelling the development of target disadvantages as sequences of events. On the other hand, it touches upon the natural limitation of regular CECA diagram by modelling essentially different aspects (conditions and actions) with same type of objects and asserting additional constraints regarding structure of the cause-effect chains to counterbalance that drawback. This construction looks artificial, compared to the original method, where all boxes are meant as disadvantages.

This paper reframes Yoon's approach by pointing out that cause-effect analysis actually identifies harmful processes responsible for "producing" target disadvantages, which may be represented using the state machine model. We have proposed rules for converting a CECA diagram into a state machine diagram and indicated required extensions to basic state machine definition. The resulting model employs heterogenic information about actions, conditions, logical operators and their interconnections inherited from the input cause-effect diagram. Such representation offers several benefits:

- stages of the process and transitions between stages are clearly distinguished, 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,
- states and transitions appear to be more intuitive representations for behavior of a process than the regular CECA diagram (which looks static in comparison),
- state machine approach is well known in the IT domain and other areas, which increases chances of a successful communication with specialists in these areas,
- existing state machine notations facilitate automatic processing of model description (e.g. syntactic validation or verification of properties),
- information about objects, functions, parameters and threshold values of parameters contributing to development of the target disadvantages may be easily extracted from the state machine model described in a standard notation,
- finally, simplification and unification of the model building blocks seems to follow the Trend of Increasing Degree of Trimming [13].

Starting this research, we expected to obtain results similar to previously achieved with the combinatorial logical model reflecting structure of the CECA diagram [11, 12]. Likewise, we aimed at developing a sequential logical model, which would allow for content-independent analysis using methods coming from Boolean algebra, such as minimization, refactoring, race detection etc. This attempt failed, because we were not able to devise an appropriate separation of model layers. Instead, a content-aware state machine approach was proposed for modelling of causality diagrams, with little so far – albeit promising – outcome. Current results need verification and enhancements, because some elements seem rough, such as different approaches to modelling OR and AND operators and counterintuitive concept of many active states, to name a few.

Further work could address transformations (e.g. reduction) of the resulting diagram. It is also known, that every state machine defines a formal grammar describing rules of creating input expressions (sequences of input symbols) capable of causing transitions to final states. This duality indicates an interesting direction of research: analyzing causal relations using linguistic approach. Another area of further work is identifying and exploring possible connections or fusions with other methods of cause-effect analysis, with Interaction Causality Scheme [17] and vulnerability-based approach [18] in the first places.

Acknowledgments. Author gratefully acknowledges Dr. Oleg Abramov for valuable materials and explanations regarding the CECA method, Mr. Piotr Salata for inspiring discussions and Mr. Dariusz Burzyński for helping to make the paper comprehensible.

References

1. Litvin, S.S., Akselrod, B.M.: Cause-Effects Chains of Undesired Effects, Methodical Theses (in Russian). CPB, 1995/12/18 – 1996/01/03.
2. Abramov, O., Kislov, A.: Cause-Effect Analysis of Engineering System's Disadvantages, Handbook on Methodology (in Russian), Algorithm, Ltd., 2000.

3. Falkov, D.S., Misyuchenko, I.L.: Analysis of typical errors made when choosing logical functions (in Russian). 2013. http://www.metodolog.ru/node/1643, last accessed 2018/07/10.
4. Falkov, D.S., Misyuchenko, I.L.: Characteristics of building fragments of Cause-Effect Chains with serial connection of Disadvantages (in Russian). 2013. http://www.metodolog.ru/node/1654, last accessed 2018/07/10.
5. Efimov, A.V.: Identification of Key Disadvantages and Key Problems using Cause-Effect Chains of Undesired Effects (in Russian). 2011. http://www.metodolog.ru/node/993, last accessed 2018/07/10.
6. Pinyayev, A.M.: A Method for Inventive Problem Analysis and Solution Based On WhyWhy Analysis and Functional Clues. TRIZ Master Thesis; 2007.
7. Medvedev, A.V.: Algorithm for Automated Building of Cause-Effect Chains of Disadvantages (in Russian). TRIZ Master Thesis; 2013.
1. Souchkov, V.V.: A Guide to Root Conflict Analysis (RCA+). ICG Training & Consulting. http://www.xtriz.com/publications/RCA_Plus_July2011.pdf, last accessed 2018/07/10.
8. 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. 184194. MATRIZ (2014).
9. Lok, A.: A Simple Way to Perform CECA And Generate Ideas in Practice. In: Souchkov, V. (ed.) Proceedings of the TRIZfest2017 International Conference. Krakow, Poland, pp. 23-30. MATRIZ (2017).
2. Chrząszcz, J., Salata, P.: Cause-Effect Chains Analysis using Boolean algebra. TRIZ Future 2016 Conference. Wroclaw, Poland, 2016. To be published in: Koziołek, S., Chechurin, L., Collan, M. (eds.) Advances and Impacts of the Theory of Inventive Problem Solving. The TRIZ Methodology, Tools and Case Studies, Springer (2019).
10. Chrząszcz, J.: Quantitative approach to Cause-Effect Chains Analysis. In: Souchkov, V. (ed.) Proceedings of the TRIZfest2017 International Conference. Krakow, Poland, pp. 341-352. MATRIZ (2017).
11. Ikovenko, S.: Level 1 Certification TRIZ Workshop, pp. 171-193. MATRIZ (2016).
12. Russo, D., Duci, S.: How to Exploit Standard Solutions in Problem Definition. Procedia Engineering 131 (2015), pp. 951-962. doi: 10.1016/j.proeng.2015.12.407.
13. Wagner, F., Wolstenholme, P.: Misunderstandings about state machines. Computing & Control Engineering Journal, vol. 15, no. 4, pp. 40-45, Aug.-Sept. 2004.
14. 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 2018/07/10.
15. Axelrod, B.: Systems approach: modeling engineering systems using Interactions Causality Scheme. In: Grundlach, K. (ed.) Proceedings of TRIZ Future 2007 Conference. Frankfurt, Germany (2007), pp.131-138.
16. Chrząszcz, J.: Indicating system vulnerabilities within CECA model. To be presented during TRIZfest-2018 International Conference, Lisbon, Portugal (2018).