VTT Technical Research Centre of Finland

Change-based causes in counterexample explanation for model checking

Ovsiannikova, Polina; Pakonen, Antti; Vyatkin, Valeriy

Published in:
IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society

DOI:
10.1109/IECON48115.2021.9589122

Published: 16/10/2021

Document Version
Peer reviewed version

Change-based causes in counterexample explanation for model checking

Polina Ovsiannikova*†, Antti Pakonen‡, and Valeriy Vyatkin*†§
*Computer Technologies Laboratory, ITMO University, Saint Petersburg, Russia
†Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland
§Department of Computer Science, Computer and Space Engineering, Luleå Tekniska Universitet, Sweden
‡VTT Technical Research Centre of Finland Ltd., Espoo, Finland
Email: polina.ovsiannikova@aalto.fi, antti.pakonen@vtt.fi, valeriy.vyatkin@aalto.fi

***Abstract*—Formal verification by means of model checking avails in discovering design issues of safety systems at the early stages. However, a significant amount of time and effort is required to decipher its results and localize the failure, especially in complex logic. This work continues our previous study on the visual explanation of failure traces and introduces change-based causes. Additionally, inspired by the types of properties that revealed model failures in projects of VTT in the Finnish nuclear industry, we define a new form of explanation – a hybrid influence graph. The new approach was implemented in a tool called Oeritte and evaluated using two practical examples of failures in nuclear instrumentation and control systems.**

***Index Terms*—counterexample explanation, user-friendly model checking, causality, function block diagrams**

I. Introduction

This work is a continuation of our previous investigation [1] of causality in counterexample explanation for model checking [2]. The problem arose from the fact that model checking being a powerful instrument for formal model verification lacks user-friendliness in the representation of its results in case the model behavior is incorrect. In brief, to model check a system, one needs (1) to obtain its formal model, (2) to create its specification using formal languages, and (3) to provide it as input to a tool, model checker. The output of the latter is then the answer "Yes" if the specification holds, and a counterexample otherwise. *A counterexample* of length l, in turn, is a sequence of system states, each of which is a set of all variables of the system with their values. Such sequence can be finite or infinite, in the second case, it consists of a finite prefix and a loop. A counterexample, generally, is a model trace where, at some point, the system property violates, however, being a table of values, it does not avail in subsequent model debugging.

In the previous work, we reduced the problem of counterexample explanation on a function block diagram (FBD) to a search for a union of all the inclusion minimal causes (IMCs) for a value of a variable at a particular counterexample step (an assignment). Intuitively, our IMC is a minimal set of assignments that are required to infer the explanation target after a finite number of steps of logical inference in the direction of the information flow. An example of such an explanation is shown in Figure 1[1].

The practice showed that the explanation in form of all the IMCs reduces the diagram area to be analyzed, however, it still might occur too crowded with highlighting and include the assignments that are not connected to the actual issue. Therefore, in this paper, we present an approach that complements the previous one and aids the analyst to focus on the variables that changed their values and influenced the verification result. Take as an example a specification formulated in terms of linear temporal logic (LTL). It can hold on some finite prefix of length m of a counterexample but turn false in the state $m+1$ ($m < l$). Showing such changes gives an analyst immediate clues about the dependencies between variables and points to the assignments, which, had they different values, the fault would have been avoided. It is especially useful in process of model debugging, as we noticed that when it is not a major design fault, the crucial element can be found locally by searching for a minimal set of assignments that should be changed in order to obtain the expected value of the explanation target. Therefore, as the main contribution of the current work, we (a) formulated a new kind of a cause – a change-based cause, (b) incorporated it in a new kind of explanation – a hybrid explanation graph, and (c) implemented it in our tool, Oeritte.

II. Previous work and inclusion minimal causes

In [1], we proposed a definition of *inclusion minimal causes (IMC)*. Here, we briefly revisit the key concepts and explain the definition from the intuitive point of view. For the

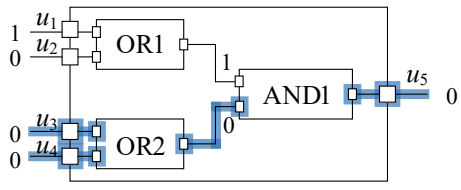

Fig. 1: Blue highlighting corresponds to explanation of the value of u_5 at counterexample step n with IMCs. The union of its IMCs include values of the following variables at step n: u_3 and u_4 (inputs of the FBD), two inputs and the output of `OR2`, the input and the output of `AND1`, and u_5 itself.

[1]Here and in further figures `0` and `1` correspond to `false` and `true`.

This work was supported, in part, by the Finnish Research Programme on Nuclear Power Plant Safety 2018-2022 (SAFIR 2022) and by the Government of the Russian Federation under Grant 08-08.

formal definitions, we invite the reader to examine Section II.A and Section III of [1].

In this paper, as well as in [1], our models are *function block diagrams (FBD)*[2]. They consist of *function blocks* (or *blocks*) which compute values of their output variables based on values of their input and internal variables. We denote input and output variables as *interface variables*, which can be connected to each other, thus, having their values equal. Output variables may have any finite number of outgoing connections, while input variables are bounded with one incoming connection at most. We say that two blocks are connected if there exists at least one connection between their interface variables.

Function blocks are divided into *atomic* and *modular*. The first kind represents atomic operators or simple functions. In this work, we use the same set of atomic block types as in [1]. A modular block, in turn, internally, is a net of interconnected blocks of any type, decomposable into a net of a finite number of atomic blocks. Therefore, an FBD itself is a modular block that is not nested into any other modular block.

Our models are discrete-time and, on each time instant, values of all the variables included in an FBD are updated. The integer *time step* is not a variable directly accessible in the model without modifications but is defined on the model execution sequence as a number of an execution. The execution semantic of an FBD is synchronous, the output values of each block are functions of its input values from the current or the previous time step (custom initialization may be applied on the first time step, and input variables without incoming connections are assigned the same default values at all time steps) and the signals are propagated through connections instantly. Delay function blocks help to prevent infinitely fast information flow in FBDs with feedback loops. Such blocks delay the signal by a single time step and each FBD may have a finite number of delay blocks.

The most basic terms of our work that require repeating their definitions are an *assignment* and a *counterexample*. Assume having FBD D with its finite set of variables $U = \{u_1, ..., u_n\}$.

Definition II.1 (Assignment)**.** An *assignment* a is a tuple $(u, v_{u,j}, j)$, where $v_{u,j}$ is the value of variable u at discrete time step j. By $v(a)$ we denote the value of this assignment and by $s(a)$ its step. If $u \in U$ is a variable of D then there exists an index $i \in [1, n]$ for u, and we denote the assignment of $u = u_i$ at time step j as $a_{i,j}$.

Definition II.2 (Counterexample)**.** A *counterexample* X of length l is a set of assignments of the variables from U for each time step j: $X = \{(u_i, v_{i,j}, j) \mid i \in [1, n], j \in [1, l]\}$.

Essentially, X is a sequence of model states (or the sets of values of all the variables at the corresponding time steps).

Now, our initial task is to explain the false outcome of an LTL formula on counterexample X of FBD D using its blocks and variables. However, as soon as the formula failure might be explained with a set of assignments [3], [4] of formula variables, we reduce this task to the explanation of a single assignment from X in D and call such an assignment an *explanation target*.

Each function block infers its outputs based on the defined rules, therefore, imposing logical *constraints* on the values of its variables. Also, each connection can be described as a constraint on the values of two variables. Suppose that each assignment is a statement. Then, it is possible to define a *cause* intuitively as a set of statements $C \subseteq X$ which is sufficient to infer the explanation target if the allowed rules are limited to using constraints of each individual atomic block or connection in the direction of the information flow. If there is no other subset of C satisfying mentioned condition, then C is an *inclusion-minimal cause*. Therefore, our *explanation* is a union of all IMCs of the explanation target.

The algorithm that finds all IMCs of an assignment was implemented in the graphic tool, Oeritte[3], that visualizes counterexamples in an FBD and explains them (1) using only variables of an LTL formula and (2) using the model as a whole.

III. Changes-based explanation

In this section, we provide an additional view on the same problem of counterexample explanation reduced to the explanation of a single assignment (or an explanation target t) from X on D and reshape our explanation from [1].

A. Explanation as a graph

Earlier we stated that our explanation is a set of assignments, which, even though can be displayed as a tree in our counterexample explanation tool, Oeritte, essentially, are not interconnected. However, due to the fact that the definition of an IMC is based on inference, each assignment in the explanation, except for the explanation target, is included in an IMC of some assignment in the explanation. Also, as multiple outgoing connections are possible in an FBD, one assignment can be included in several IMCs.

This brings us to a conclusion that the assignments from the union of IMCs of the explanation target can be organized into a directed *IMC influence graph*, where nodes are assignments and edges are ordered pairs (a, a'), where a, a' are such assignments that a' is included into the local IMC of a. For further convenience we will define an *influence graph* in a generic way, i.e., its nodes are $A \subseteq X$ and edges are ordered pairs of assignments from A connected with particular relation. The graph will be named according to this relation, e.g., in this case, IMC influence graph.

Therefore our explanation now is not just a set of assignments but an IMC influence graph in which a set of nodes correspond to the union of IMCs of t.

B. Change-based causes

As a verification result, we might get a scenario where the specification is satisfied on some finite prefix of the counterexample but starts failing if we include the next step to

[2]Our approach is not limited to the programming language called FBD in the IEC 61131-3 standard. We use "FBD" in a more general sense.

[3]https://github.com/ShakeAnApple/cxbacktracker/

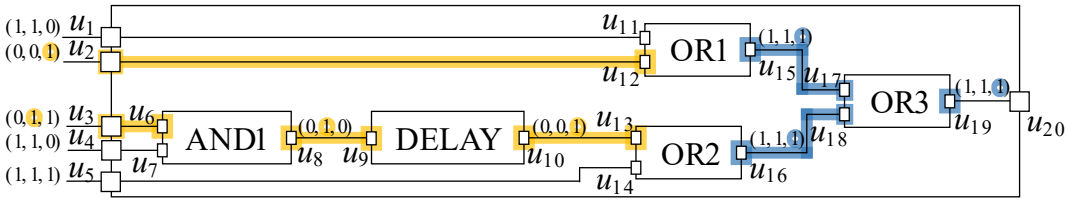

Fig. 2: Example of a hybrid explanation visualized in an FBD. Here, we assume that the counterexample consists of three states. Triples next to the names of the variables or above the connections indicate values of the corresponding variables at each step starting with step one if reading from left to right. The change-based explanation is marked with orange highlighting, while blue corresponds to inclusion-minimal causes explanation from [1].

the trace. This might happen because some variables changed or because variables, that should have changed, preserved their values (or if we deal with numeric variables, changed in an inappropriate way). In the first situation, it is important to direct the attention of the analyst to the specific process that prevented the property to be fulfilled. Therefore, we define a new type of cause, a *changed-based cause* that is defined through a *local changed-based cause*.

Definition III.1 (Local change-based cause)**.** A *local change-based cause* $\widetilde{C}^* \subseteq X$ of t is such a set of assignments that is included into the union of local IMCs of t and $\forall a_{i,j} \in \widetilde{C}^* : v(a_{i,j}) \neq v(a_{i,j-1})$.

Our local change-based cause, generally, will include a subset of assignments of inputs of an atomic block or an assignment if the opposite side of a connection, depending on if the target is an output or an input. This subset contains not only assignments that have changed at the last step (with respect to t) but which also influence the value of t as they are chosen from its IMCs. Consider an FBD in Figure 2. A local change-based cause of an output u_{10} at step 3 here is a singleton containing assignment $(u_9, 1, 2)$. As soon as u_9 is always assigned the value of u_8, a local change-based cause of $a_{9,2}$ is the same as of $a_{8,2}$ which is $\{(u_6, 1, 2)\}$. Now, as we know all the variables of the system and there are no hidden processes that may influence their values, we can say that a change in the value of one variable cannot happen without premises, i.e., changes in the values of some other variables, which, in turn, also require premises. Thus, we can expand the definition beyond the local explanation scope and formulate a *change-based cause*.

Definition III.2 (Change-based cause (CBC))**.** A set of assignments $C^* \subseteq X$ is a *change-based cause (CBC)* of a target t if there exists such sequence of sets of assignments from X, $Y_0, ..., Y_m : C^* = Y_m, t \in Y_0$, where each $Y_{k+1}, k \in [0, m-1]$ extends Y_k with one or more local change-based causes of any assignments from Y_k.

This means that local CBCs are also CBCs. Intuitively, we say that if assignment a is a local CBC of t, and assignment a' is a local CBC of a, then a' is also a CBC of t, or, as in our example in Figure 2, $C^* = \{(u_6, 1, 2)\}$ is a CBC of $a_{10,3}$.

As a result, our change-based explanation is a CBC influence graph where the set of nodes corresponds to a union of all CBC of t and edges assignments are connected with a change-based causal relation.

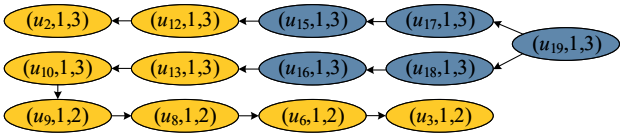

Fig. 3: Explanation (hybrid influence graph) for assignment $(u_{19}, 1, 3)$ from the FBD in Figure 2. The ovals with blue and orange background correspond to assignments included into IMCs and CBCs respectively.

IV. Hybrid Explanation

Change-based causes effectively point to critical processes taking place in a model, however, their usage is limited to failure scenarios where the satisfactory system operation is interrupted by the unexpected deviation. In this case, such instant changes draw the attention of the analyst to the diagram areas which should not allow the combinations of values found, or to the fact that the LTL property checked should be reformulated if the counterexample is spurious.

On the other hand, this work was inspired by the practical cases of applying model checking in the Finnish nuclear industry and is aimed to avail in explanation of failures of the most common types of LTL properties in this domain. In [5], the authors collected and categorized specifications that were of interest throughout a two-year experience of VTT in the nuclear industry. Out of 1079 properties in total, 87% were formulated using LTL and 92% can be described as implication with a leading type $\mathbf{G}(p \rightarrow q)$. Such a formula fails if anywhere across the state space of a model statement $p \wedge \neg q$ is `true` and this failure, for example, might mean that despite some criteria was satisfied, the expected outcome was not obtained.

Investigating counterexamples for properties of this type, we found out that the most usual failure scenario that a model checker discovers comprises of a prefix where both p and q are `false` followed by a state where p becomes `true` but q does not change. Considering that q commonly represents a predicate formulated using output or internal variables of the model verified, the question of the analyst interest here is why q remains `false` while it should change.

Consider the case where q is a predicate of a single variable. Here, CBC helps to reveal a situation when this variable *stayed unchanged despite other changes taken place* and indicates that some concurrent event happened that canceled the effect of the change. But there also exists a scenario where *none of the variables*, whose assignments are included in the union of local IMCs of t, *changed* from the start of a counterexample

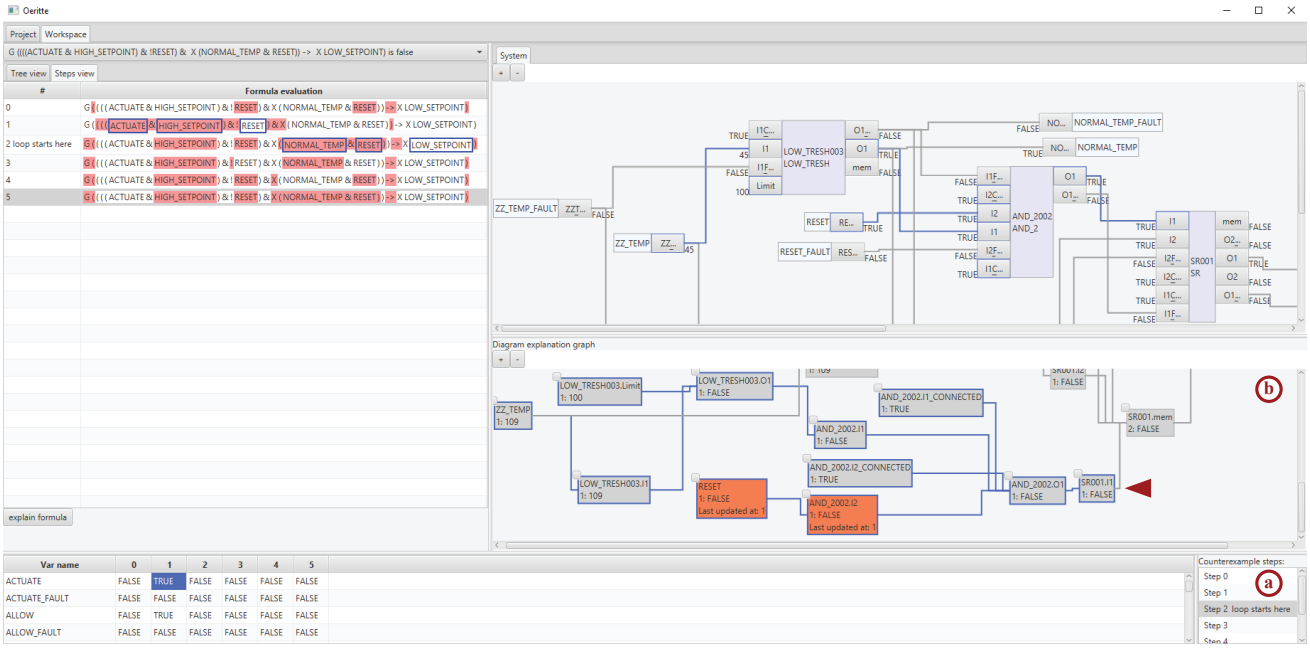

Fig. 4: Main view of Oeritte, a tool for visual counterexample explanation. Blue selection in diagram (b) highlights nodes of a hybrid influence graph that are reachable from the chosen node, which is, in this case, the rightmost highlighted node, pointed with the red triangle (not a part of the tool user interface), and edges between them.

till the failure step, which leaves the analyst empty-handed. This fact motivated us to search for a hybrid explanation approach that includes both change-based and inclusion-minimal causes.

A. Explanation approach

In our hybrid explanation of t, we synthesize explanation approaches based on CBSs and IMCs. Therefore, our explanation is a hybrid influence graph, where (1) edges assignments are connected with inclusion minimal or change-based causal relation, (2) each assignment node $a_{i,j}$ is included into the union of all IMCs of t and if $v(a_{i,j} \neq v(a_{i,j-1}))$ then $a_{i,j}$ is also included into the union of all CBCs of t and (3) paths from assignments from CBCs can end only in assignments from CBCs. Thus, unless an assignment a has a local CBC, it is connected to the assignments from its union of local IMCs, otherwise, a is further explained only through changes. Figure 2 visualizes on a diagram the hybrid explanation of the value of u_{19} at counterexample step 3, provided in Figure 3.

To infer such a graph we need an adjusted version of Algorithm 1 from [1]. More specifically, now, after finding the IMCs for an output of an atomic block in line 7, we check if some of the result assignments have changed since the previous step. If yes, then we only append them to the result, otherwise, we append all the found assignments. Besides, as now an explanation is defined as a graph, we also create and update the list of its edges.

B. Implementation in Oeritte

Since [1] the user interface of Oeritte has been slightly changed (Figure 4). We removed the diagram and formula explanation result panels and moved the clickable counterexample steps list (Figure 4 (a)) to the lower area of the main view. In the freed-up space, we built the control which displays a hybrid influence graph for a chosen assignment (Figure 4 (b)).

To address the need to learn why the variable of the explanation target remained unchanged during several counterexample steps, we decided to implement an experimental feature and run an explanation procedure for each of such steps. The result graphs for each of the steps are then merged by the same assignments and displayed in the graph control. In this control, gray and orange boxes are the assignments included into causes, IMC or CBC respectfully, gray and blue lines represent edges. Each assignment has a small button in the top left corner, click on which highlights all the paths in the graph from the current node with blue. For nodes that were changed at some point in the past but not at the previous step, we display when the last change took place.

V. Case Study

For the evaluation of our new approach, we use two examples from our previous works and types of LTL properties that are often verified in nuclear industry projects in VTT [4], [7].

A. PID control of a safety actuator

As our first practical example, we use a PID control of a safety actuator and its design issue from [6], Figure 6 presents its slightly modified FBD. Here, `MEASURE` is defined based on second-minimum voting over four redundant signals. In the control mode, this value is memorized and then used as a `SETPOINT` for the controller. In case of measurement fault or when the control mode is switched off, the controller is disabled.

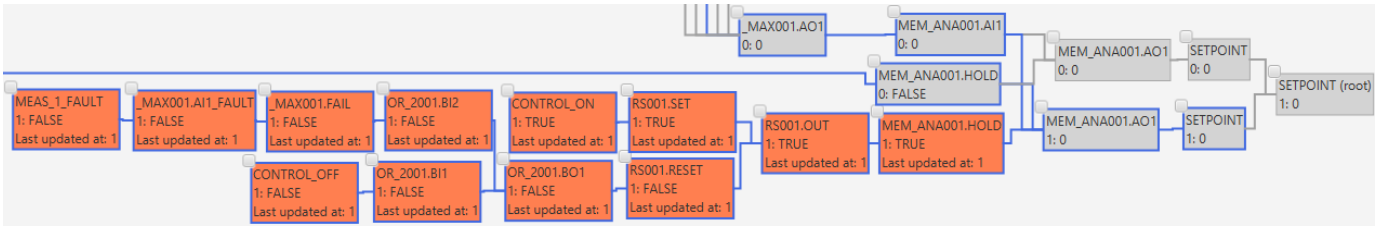

Fig. 5: Part of the hybrid explanation graph for the design issue from [6] for the system in Figure 6. Here we can see that the explanation for variable `SETPOINT` at step 1 includes explanation for `RS001.OUT` at the same step and they both are included in a cause of an LTL formula failure.

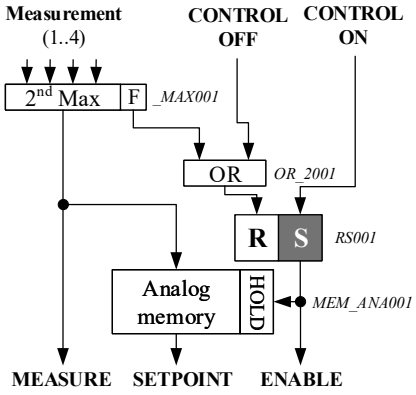

Fig. 6: PID control logic example (the PID controller itself, was omitted to save space). Block [R, S] stands for a memory with reset priority. We put names of the blocks in NuSMV model in *italic* to the right of the blocks.

We were interested in the same property as in [6], i.e., "assuming that the valid (actual) measurement is never zero, then zero shall never be selected as the setpoint", which is equivalent to LTL formula "$\mathbf{G}\,\neg(\texttt{MEASUREMENT} = 0 \wedge \neg\texttt{MEASUREMENT\_FAULT}) \rightarrow \mathbf{G}\,\neg(\texttt{SETPOINT} = 0 \wedge \texttt{ENABLE})$". This property was proved to be false by the NuSMV [8] model checker and Oeritte showed that the formula failed at the second counterexample step, where the right part of the implication became `true`. This happened because the value of `SETPOINT` was equal to 0 and the controller was enabled at the same time. Checking the explanations for both of these variables we notice that the one for the `SETPOINT` at step 1 (the tool numbers steps starting 0) includes the explanation for `ENABLE` (which is equal to `RS001.OUT`) at the same step (Figure 7). Here, *the change that happened in the system causes the important variable to preserve its value*. The explanation graph shows that `ENABLE` was set to `true` because the control mode was switched on at step 1, which made the analog memory preserve its value that was set to 0 at the previous step (step 0, the initial state). In other words, due to measurement fault at step 0, the second-maximum block outputs a default preset initial value of `0`, not a value based on real measurements. This value is then memorized as the set point.

B. Set point selection logic

In our second experimental evaluation, we use a set point selection logic that was described in [7]. Here we analyze the same simplified and masked version of the original design and its issue. A diagram of the logic—slightly modified from [7]—is provided in Figure 8.

We formulated the model in terms of NuSMV verifier and, as in [7], checked that "When the temperature has returned to normal level, the operator can reset the set point to normal value", which was represented in LTL as "$\mathbf{G}(((\texttt{ACTUATE} \wedge \texttt{HIGH\_SETPOINT} \wedge \neg\texttt{RESET}) \wedge \mathbf{X}(\texttt{NORMAL\_TEMP} \wedge \texttt{RESET})) \rightarrow \mathbf{X}\,\texttt{LOW\_SETPOINT})$". The model checker produced a counterexample, where the left part of the implication became true at step 1 but the value of `LOW_SETPOINT` remained `false` at the next step. The part of the hybrid influence graph produced by Oeritte is shown in Figure 7. Here we can see that the value of `LOW_SETPOINT` remained `false` *despite the changes* happened in the system. More precisely, output of `LOW_TRESH003` (which corresponds to `NORMAL_TEMP`) together with `RESET` became `true` at step 2. As they both are inputs of `AND_2002`, the output of the latter also became `true` and set the output of flip-flop `SR001`, which, in turn, being received by `AND_2001` inverted, set its value to `false`. However, as the graph points us, `AND_2001` did not have a chance to result in `true` as its another input was set to `false` by the `ACTUATE` signal that also became `false` at step 2. This `false` signal was propagated to `AND_2008` whose output corresponds to `LOW_SETPOINT`. The analyst infers that the issue here is that the operator uses the reset option while the temperature is still high.

VI. Discussion and Related Work

The definition of a change-based cause in this work can be considered as counterfactual [9], as it follows the logic "Unless event A had happened, event B would not have happened", where events A and B, in our case, are particular assignments. However, despite CBCs avail in exploring fault propagation paths, we found pure counterfactual definition insufficient to form an explanation and developed a hybrid approach that includes the general type of causes from the previous work.

Our novel hybrid approach not only reduces the explanation but also reveals the reasons behind the situations when a variable preserved its value despite the change performed or when it changed contrary to what was expected. The synthesis of two explanation methods, change-based and inclusion-minimal, was especially highlighted in our first case study example in Section V-A, where the value of `ENABLED` could be explained only with CBCs, while `SETPOINT` required inclusion-minimal explanation at the beginning.

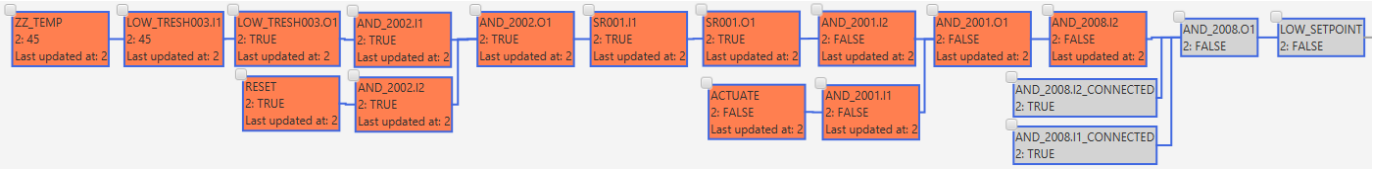

Fig. 7: Part of the hybrid explanation graph for the design issue from [7] for the system in Figure 8. The problem here is that `LOW_SETPOINT` remained `false` at step 2. This happened because values of some input variables were changed in a way that another input of `AND_2008` became `false` and was propagated to `LOW_SETPOINT`.

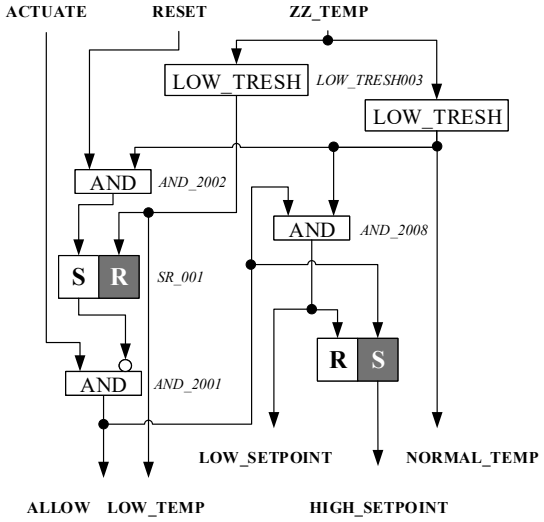

Fig. 8: Set point selection logic example. Blocks $[\mathtt{S}, \mathtt{R}]$ and $[\mathtt{R}, \mathtt{S}]$ correspond to memory with set and reset priority, a circle between the arrow and the block means that this input is inverted. We put names of the blocks which are important in the explanation in *italic* to the right of the blocks.

To save space, we do not provide a full literature review on the current problem since it was performed in our most recent work. However, we mention the closest works to ours which are [3], [10] in the scope of explanation using specification and [11]–[13] in explanation using the model itself. The closest related approach is still [14], from which we differ now also in our new change-based causes.

VII. Conclusion

In the presented work we developed a new approach to the explanation of the violated LTL properties. This task was reduced to the explanation of a single assignment using CBCs and IMCs. To evaluate the new approach, we implemented it in Oeritte and explained two design issues taken from the practice of VTT in the Finnish nuclear industry.

In our implementation, we not only display immediate CBCs and IMCs but also show when each variable included in the explanation changed its value the last time. Also, when the user requires the explanation of the variable that preserved its value during several counterexample steps, we show its explanation for all the previous steps where the value remained. Nevertheless, the last feature requires more computational resources and, presumably, a more compact representation of the influence graph to fully assist the user.

Another substantial remark is that the explanation is more beneficial if the system model supports user-friendly design (e.g, NuSMV modules are named after their functions, variables names follow a single convention).

Our future work is aimed at improving the user experience with the influence graph and reducing the computational resources required to perform the explanation of variables that preserve their values.

References

[1] P. Ovsiannikova, I. Buzhinsky, A. Pakonen, and V. Vyatkin, "Oeritte: User-friendly counterexample explanation for model checking," *IEEE Access*, vol. 9, pp. 61 383–61 397, 2021.

[2] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, Cambridge, Massachusetts, 1999.

[3] I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. Trefler, "Explaining counterexamples using causality," *Formal Methods in System Design*, vol. 40, no. 1, pp. 20–40, 2012.

[4] A. Pakonen, I. Buzhinsky, and V. Vyatkin, "Counterexample visualization and explanation for function block diagrams," in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, 2018, pp. 747–753.

[5] A. Pakonen, C. Pang, I. Buzhinsky, and V. Vyatkin, "User-friendly formal specification languages – conclusions drawn from industrial experience on model checking," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–8.

[6] A. Pakonen, I. Buzhinsky, and K. Björkman, "Model checking reveals design issues leading to spurious actuation of nuclear instrumentation and control systems," *Reliability Engineering & System Safety*, vol. 205, p. 107237, 2021.

[7] A. Pakonen, "Model-checking of I&C logics – insights from over a decade of projects in Finland," in *12th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies (NPIC & HMIT)*. American Nuclear Society, 2021.

[8] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An OpenSource tool for symbolic model checking," in *International Conference on Computer Aided Verification (CAV)*. Springer, 2002, pp. 359–364.

[9] D. Lewis, "Counterfactuals and comparative possibility," in *Journal of Philosophical Logic*. Springer, 1973, vol. 2, pp. 418–446.

[10] A. Ek, "Explanation of counterexamples in the context of formal verification, Bachelor's thesis, Uppsala University, Department of Information Technology," 2016.

[11] A. Groce, D. Kroening, and F. Lerda, "Understanding counterexamples with explain," in *International Conference on Computer Aided Verification*. Springer, 2004, pp. 453–456.

[12] F. Leitner-Fischer and S. Leue, "Causality checking for complex system models," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 2013, pp. 248–267.

[13] C. Wang, Z. Yang, F. Ivančić, and A. Gupta, "Whodunit? causal analysis for counterexamples," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2006, pp. 82–95.

[14] T. Bochot, P. Virelizier, H. Waeselynck, and V. Wiels, "Paths to property violation: A structural approach for analyzing counter-examples," in *2010 IEEE 12th International Symposium on High Assurance Systems Engineering*, 2010, pp. 74–83.