Rigorous Automated Verification of Protection Systems in LV Distribution Grids

Ahmed Nagy Abdelkhalek Mansour, Samuele Grillo
Dipartimento di Elettronica, Informazione e Bioningegneria
Politecnico di Milano
I-20133 Milan, Italy
{ahmednagy.mansour; samuele.grillo}@polimi.it

Enrico Ragaini
ABB S.p.A
SACE
Bergamo, Italy
enrico.ragaini@it.abb.com

Matteo Rossi
Dipartimento di Meccanica
Politecnico di Milano
I-20133, Milan, Italy
matteo.rossi@polimi.it

***Abstract*—In power systems protection, selectivity is a key property that must be guaranteed. It ensures the minimum amount of load is unfed when a fault occurs and it reduces the time needed to restore service. Selectivity is achieved through coordination of circuit breakers (CBs), whose logic is most commonly determined by a combination of time and current thresholds. Verifying the correctness of CBs' settings is crucial when designing power system protections and may become a challenging task for big networks. This paper proposes an approach for the rigorous verification of the correct configuration of protection systems in low-voltage (LV) distribution grids. The method utilized depends on a precise model that employs Timed Automata (TA) to represent the crucial elements of an LV distribution grid. Additionally, it incorporates a system for producing and validating formal models from higher-level JSON-based depictions of electrical networks, taking advantage of the UPPAAL model checker. The effectiveness of the approach has been tested on several realistic power systems.**

***Index Terms*—Formal verification, selectivity, timed automata.**

I. INTRODUCTION

The protection of power systems is a critical task in ensuring the safe and reliable operation of electrical power networks [1], [2]. Protection schemes are designed to isolate faults in the system in a selective and efficient manner. Selectivity is essential because it helps to minimize the impact of faults on the power system and to reduce the time required for restoration. However, designing a protection scheme that is selective and reliable can be a challenging task due to the complexity and size of power systems [3].

Formal verification is a powerful tool for analyzing the behavior of complex systems, and for checking their correctness. It has several advantages over traditional testing and simulation methods, such as the ability to provide rigorous analysis and identify potential design flaws. In this paper, we present a formal verification approach to verify the selectivity of protection schemes in power systems. Our methodology uses formal verification techniques, and in particular model checking for Timed Automata (TA) [4], to analyze the settings of a set of circuit breakers (CBs), and to prove whether they meet the design specifications for what concerns selectivity. In order to validate our methodology, we applied it to realistic case studies.

Formal verification methods have been applied in the past to problems somewhat related to power systems. However, to the best of our knowledge, they have never been used to model the behavior of CBs and to check selectivity. The methodology presented by the authors in [5] employs formal verification to confirm the design of a photovoltaic system and, specifically, to ensure that its size is validated prior to project construction, in order to avoid the needless acquisition of equipment. In contrast, the authors of [6] adopt Satisfiability Modulo Theories (SMT) techniques to investigate a relay interlocking system, modeling basic electrical circuits and switched multi-domain Kirchhoff networks.

In [7] a probabilistic-based verification approach of directional over-current relays is presented. It is based on probabilistic model checking and provides a Markovian model of directional relays. These models are used to verify relays' settings and operational behavior, which allows users to obtain the probability of the success or failure of certain elements of the protection system. However, nothing is said about selectivity.

Similar to the method introduced in this paper, authors in [8] also utilize the UPPAAL model checker to verify fault location, isolation, and service restoration (FLISR) systems. Nevertheless, the approach outlined does not incorporate a model of the network topology and does not address selectivity.

Following the requirements of ABB[1], in this work we focused on the most common and basic behavior of CBs, i.e., the combination of time delays and current thresholds.

The main contributions of the paper are:

i) formal models, based on TA, are introduced, capturing the relevant behavior of CBs and protection systems;
ii) a set of queries that enable checking the correctness of the protection system configuration (in this paper we focus on selectivity, but other properties could be studied), is defined so that an automated formal verification tool can be run on them;
iii) an automated mechanism is developed and validated that, on one hand, automatically produces, starting from high-level descriptions of the protection system and of the network, the formal models of point i) and the

The work leading to this paper has been developed under the framework provided by the Joint Research Center between ABB and Politecnico di Milano.

[1]Given the industrial nature and origin of the work, artifacts such as models and code cannot be made available to the academic community, as they could be included in ABB's tools in the future.

979-8-3503-4743-2/23/$31.00 ©2023 IEEE

queries of point ii), and on the other hand executes the formal verification tool to perform the desired checks.

With respect to point i), the proposed methodology characterizes the pertinent attributes of protection systems using TA, given that the timing of events (e.g., fault occurrence and CB tripping) is one of the key distinguishing features of the analyzed systems. Next, regarding point ii), the formal verification tasks are automatically executed using the UPPAAL model checker [4]. Ultimately, in addressing point iii), we enable users to define the critical components of protection systems (e.g., network structure and CB configuration) using JSON files and create a prototype transformation tool based on MATLAB®.

At present, the proposed technique is designed for *radial networks*. Despite being a subset of all possible electrical networks, radial networks are still significant: all low-voltage (LV) distribution networks (such as those that are connected directly to buildings and houses) are radial.

The structure of the paper is as follows: In Section II, presents essential information on the fundamental concepts of TA and formal verification. Section III introduces the proposed approach's formal model of protection systems and the corresponding queries for selectivity verification. Section IV provides an outline of the mechanisms utilized to produce TA models and queries from JSON-based descriptions, which can then be fed into the UPPAAL model checker to execute formal verification activities. In Section V, the proposed approach is tested using realistic networks, and the feasibility and functionality are evaluated. Finally, Section VI provides a conclusion.

II. BACKGROUND ON FORMAL VERIFICATION AND TA

Formal verification approaches are based on a few ingredients [9]: (i) a mathematical model of a target system to be studied; (ii) a language to express the properties to be checked for the system; (iii) a mechanism to check if the desired property holds or not for the target system. A plethora of formal verification approaches are available in literature, with different characteristics. We can separate them based on: the nature of the notation used to capture the system (e.g., based on automata, logics, process algebras); whether they allow the modeling of temporal properties ir not and, if they do, what kind of temporal model they support (discrete, continuous, metric, etc.); the language used to express the properties of interest (which is often based on logics, although the specific kind of logic varies); the level of automation of the property checking mechanism (manual, semi-automated, fully automated) and, if the approach is at least semi-automated, whether it is supported by tools or not.

In this work, we target a verification approach that: (i) allows users to capture *metric* temporal properties of the systems to be analyzed (to study the timing of the triggering of CBs); (ii) is fully automated, to make it accessible also for users who are not experts in formal verification techniques; (iii) can be integrated as core of a larger software tool that includes formal verification functions.

TA model checking is a formal verification approach that satisfies all necessary requirements. In particular, TA [10] are an automata-based formalism that allows users to precisely describe timing properties of systems in a metric manner; TA can be automatically analyzed through various tools, and in particular through the UPPAAL model checker [4].

We illustrate the features of TA through the example of Figure 1, which shows a TA defined through UPPAAL. A TA evolves from an initial *location* (e.g., `Closed` in Figure 1, as highlighted by he double circle) by taking *transitions*, which can cause the TA to change location. TA capture the advancement of time through the notion of *clock* (e.g., x in Figure 1). When a TA takes a transition, it can *reset* zero or more clocks, which keep track of the amount time that passes since their reset. In addition to clocks, a TA can include integer variables that store information of which the TA needs to keep track. Transitions can include *guards*, which are conditions on clocks and integer variables that must be true for the transition to be taken. They can also have *updates* (i.e., assignments) of integer variables, which change the value of the latter. For example, the transition from `Closed` to `Standby` resets clock x. The transition from `Standby` to `Open` has a guard, `x == t && I1 > I1_th`, which states that the transition can be taken only if the value of clock x is equal to t, and the value of integer variable `I1` is greater than constant `I1_th`. It also has an update (performed by function `update(1)`), which assigns new values to integer variables. A location can be associated with an *invariant* (e.g., `x <= t` for `Standby`), which is a condition on clocks that must remain true as long as the automaton stays in that location. Transitions can be labeled with *channels* (e.g., `CBopen` and `Reset` in Figure 1), which are used to describe the *synchronization* among TA. Given a channel `c`, the edge that *sends* signal `c` is labelled with `c!` (e.g., `CBopen!` in Figure 1) while the edge that *receives* the signal is labelled with `c?` (e.g., `Reset?`).

Multiple TA synchronizing with each other constitute a *network*. The UPPAAL model checker takes as input a network of TA and a property to be checked (expressed in logic language, as discussed in Section III-C) and determines whether the latter holds for the network or not; if the property *does not* hold, UPPAAL returns a *counterexample* (i.e., an execution of the network that violates the property), which can be inspected by the user to determine why the property fails. UPPAAL can check various kinds of properties, such as, for example, *reachability* ones, which concern whether a certain state can be reachd from the initial one.

III. FORMAL MODELING OF PROTECTIONS

The UPPAAL model is the base of the proposed approach and must capture the key elements of the electrical network—such as the location, settings and behavior of lines, buses, and CBs—that are involved in protection systems. The model also needs to capture the evolution of the currents in every line after every significant event (e.g., fault or opening of a CB).

Since the property targeted for study is selectivity, which directly depends on the behaviour of the CBs, the model includes a TA for each CB that appears in the network; in addition, another TA, called `FaultGenerator`, is introduced to describe the occurrence of short circuit events, and monitor the status of the network. On the other hand, the current that flows in each line is defined in UPPAAL through an integer variable, whose value is set by suitable update functions included in the model.

A. Modeling Circuit Breakers

Each CB in the network is captured by a separate TA. For example, Figure 1 shows the TA of the CB on line 1 of the network of Figure 2. Each CB TA has three locations (`Closed`, `Standby` and `Open`) and four transitions.

In addition, each CB TA has a set of local declarations (functions, etc.) that are CB-specific, and that capture the CB settings. Hence, the various CBs TA differ in the settings used for the transitions, as well as the local declaration.

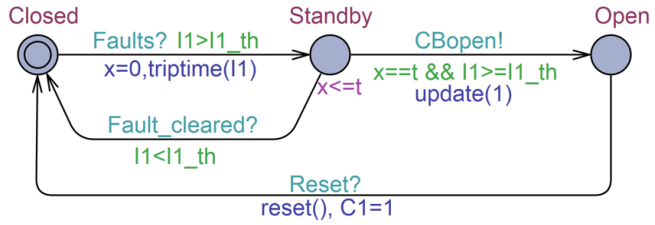

Fig. 1. TA capturing the CB on line 1 of Figure 2.

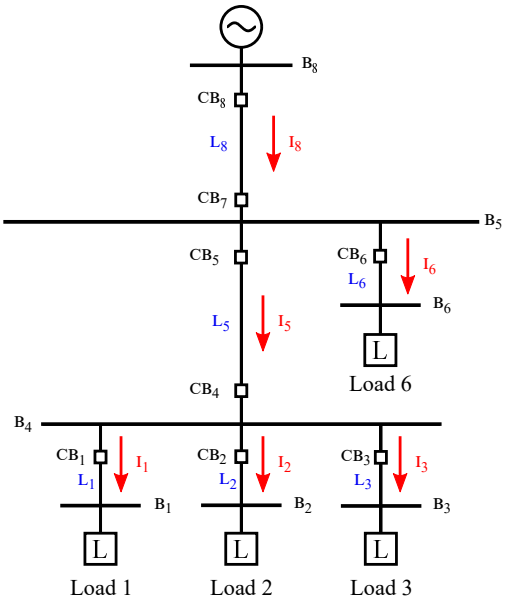

Fig. 2. Example of simple network.

When the `FaultGenerator` TA takes a transition that models the occurrence of a fault, all CB TA receive signal `Faults`, but only the CBs whose current exceeds the given threshold (in the case of the example of Figure 1, such that `I1 > I1_th` holds) actually take the transition from the initial location `Closed` to `Standby`. In other words, only the CBs that "see" the fault take the transition. When the transition is taken, the update statement resets a clock x and calls function *`triptime`* (explained in more detail below), which calculates the time t at which the CB must act if the fault has not been cleared before.

From location `Standby`, the TA switches to location `Open` if the deadline to take an action has been reached (i.e, guard `x == t` holds) and the current flowing in the CB (`I1` in this case) is still higher than the threshold (`I1_th`). When this transition is taken, signal (`CBopen`) is broadcast, notifying other TA that the CB opened, and function *`update`*,

```
int t;
int n = 3; // depends on the type of the CB,
           // 3 for Emax and 6 for Tmax
int i1 = 200; // (200 means 2000A) time at which
              // the CB will start to trip
              // (threshold current)
int i2 = 750; // short time pickup current
int t1 = 30;  // 30 means 3s, time corresponding
              // to a current = n*i1

void triptime(int I) {
     if(I < i1){
       t = 9999; // this is just conventional
     }
     else if (I >= i1 && I < i2){
       t = (n*i1*n*i1*t1)/(I*I);
     }
     else if(I >= i2){
       t = 2;
     }}
```

Listing 1. Example of *`triptime`* function.

which updates the value of all the currents in the branches based on this CB opening, is executed. The CB TA switches from `Standby` back to `Closed` when it receives signal `Fault_cleared` and the current is lower than the threshold, which happens if another CB trips first and clears the fault. Finally, if the TA is in location `Open` and it receives signal `Reset`, it moves to location `Closed` and executes function *`reset`*, which sets the values of all currents back to the initial operating conditions, thus resetting the system.

`triptime` function: This function is defined in the CB TA local declarations, which contain properties and equations that are specific to a particular CB, such as its settings. One of the commonly used approaches to achieve selectivity involves calculating delays based on the detected current, using a suitable time-current curve (such as the one in Figure 3). These curves are implemented as equations in the *`triptime`* function, which takes the current flowing through the line as input and returns the trip-time based on the CB settings. The *`triptime`* function for the CB of line 1 is provided in Listing 1. Further details about the CB settings and time-current curves can be found in [11].

B. The `FaultGenerator` TA

As its name suggests, this TA is built for the purposes of introducing faults in different parts of the network, as well as monitor the status of the network. The automaton is built in way that makes it introduce only one fault at a time and only after that fault is cleared will it introduce another fault. The automaton's details, specifically its transitions, are determined by the structure of the network under analysis. Although the TA comprises five locations that are not network-dependent, its transitions are defined based on the network configuration. Beginning from the `No_Fault` initial location, there are many possible transitions towards the `Fault_signal` location, with each transition signifying a fault occurring either on a bus or a line within the network.

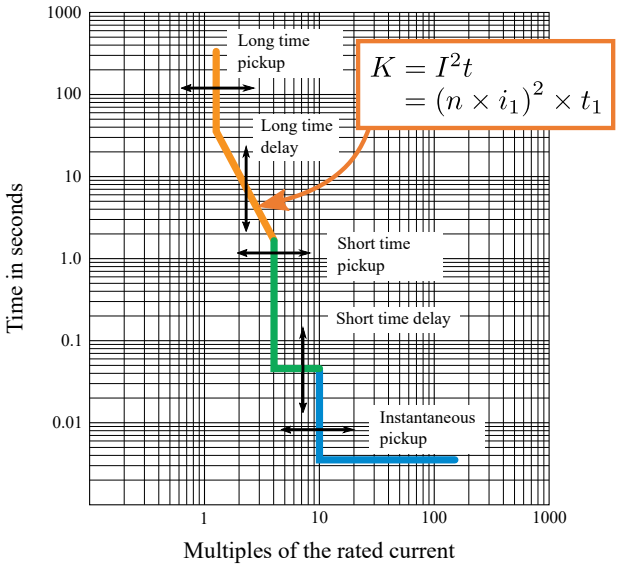

Fig. 3. Example of time-current trip curve. The curve features three distinct thresholds: i) for currents exceeding ten times the nominal current, the CB will trip "instantaneously"; ii) for currents ranging from four to ten times the nominal current, the CB will trip after a brief delay; and iii) for currents surpassing the nominal current, but not exceeding four times its value, the CB will trip following a specific time, as determined by the equation.

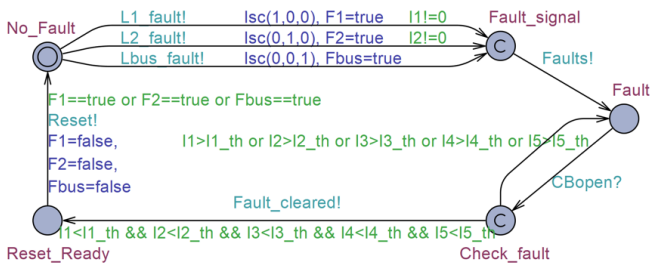

Fig. 4. Example of TA capturing the generation of faults.

The number of transitions is influenced by both the total number of buses and lines present within the system, as well as by the specific fault locations selected by the user in the configuration file (outlined further in Section IV). When one of these transitions is taken, the transition's update statement calls the function *Isc* (short-circuit function), which assigns new values for currents to all branches of the network based on the short circuit event. After reaching the location `Fault_signal`, the automaton leaves this location immediately, because it is a *committed* location [4], in Uppaal a *committed* location is a location that the TA leaves with zero time elapsed. The purpose of using such location in the `FaultGenerator` TA was to send only one signal (through event `Faults`) to all CBs TA, this made the CB TA simpler and more efficient, and the zero time elapsed property made it not affect the in any way the delays of the CBs. When the transition from `Fault_signal` to `Fault` is taken immediately after a fault occurs, and sends the message called `Faults` to all CBs, Each CB[2] reacts to this message (see Section III-A) based on the currents that were updated thanks to the *Isc* function.

The `FaultGenerator` TA stays in the location `Fault` until it receives a signal `CBopen` which is sent by any CB when it opens, this message propmts the transition to the `Check_fault` location. `Check_fault` is also a committed location, so the TA stays in it for zero time, since its only purpose is to check whether a fault is still present in the network or not. It does so through the guard statements on the next two possible transitions, where if any of the currents is still higher than the threshold current, then the transition from `Check_fault` back to `Fault` is taken because a fault is clearly still present. On the other hand, if all line currents are less than the threshold that means the fault is cleared and the automaton moves to the `Reset_Ready` location. Finally, when the transition from `Reset_Ready` to `No_Fault` is taken, it sends signal `Reset`, which causes the CB that opened in response to the fault to execute function *reset* which restores the system back to the initial operating conditions before any fault occurred, so that the system is ready to invoke a different fault(see Section III-A).

C. Capturing Selectivity in UPPAAL

As known, selectivity's goal is to minimize both the number of devices that trip and the extent of the area affected by a fault. This is achieved by coordinating the settings of the protective devices, so that those closest to the fault trip first, clearing the fault with the least disruption to the rest of the system.

Selectivity can be analyzed in UPPAAL through suitable queries capturing reachability properties (see Section II). In UPPAAL, reachability properties take the form `E<> P` (where `P` is a condition that holds in some state of the system), which means that it exists (`E`) is an execution of the network of TA such that, eventually (`<>`), property `P` holds. Consider, for example, the system depicted in Figure 2. Suppose that we wanted to check whether there is any possible scenario (i.e., an execution of the TA network) such that, while a fault on line 1 is still active, CB_5 is open, but CB_1 is not. In UPPAAL terms, the check is performed through the following query

```
E<>(CB5.Open && !CB1.Open && F1)
```

which determines whether there is any system execution such that, eventually, `CB5.Open` holds and, at the same time, `CB1.Open` *does not* hold (`!` represents negation in the UPPAAL query language), and `F1` holds (i.e., there is fault on line 1). Notice that the above property represents an *undesired* situation for the example of Figure 2, since if it were true it would mean selectivity is not achieved by the protection system and the configuration of the CB is wrong.

In the proposed approach, a list of queries is introduced for each CB, which capture whether each CB achieves selectivity with respect to all upstream CBs. The queries are automatically generated and checked by the tool supporting the proposed approach (see Section IV). If no undesired situation is detected (i.e., the results of all queries are false), we conclude that the protection system achieves selectivity; otherwise, the reachability property that is satisfied by some execution highlights the CBs that are incorrectly configured.

For example, referring to the network of Figure 2, the queries for `CB2` are the following:

[2]Notice that all channels in the UPPAAL model are *broadcast* ones.

```
E<>(!CB2.Open && CB5.Open && F2)
E<>(!CB2.Open && CB8.Open && F2)
```

IV. Automated Generation of Formal Models

Building the UPPAAL model by hand is a time-consuming process that requires expertise in both the electrical power engineering domain and the formal modelling domain, and it is prone to errors. Hence, we developed a tool that facilitates the creation and execution of the TA and queries described in Section III. This reduces both the human effort necessary to apply the presented approach and the risk of introducing mistakes in the formal models.

The tool (henceforth called the automated protection system verification (APV) tool) was developed in MATLAB® and, as depicted in Figure 5, takes as input a JSON file that contains the necessary information regarding the network structure, such as: bus and line IDs and connections, values of currents at initial operating conditions, threshold values and short circuit conditions for each line, indication of which are the buses and lines in which faults can occur, CB settings. All these data are typically readily available to designers of protection systems as part of the design process.

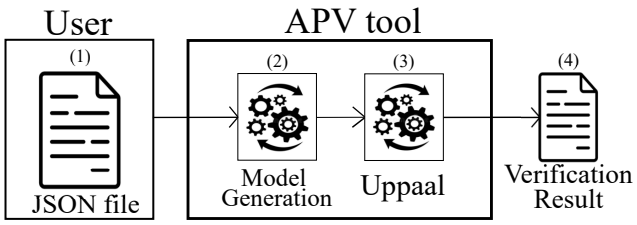

Fig. 5. Workflow of the automated model generation tool.

After the JSON file is created by the user, it is provided as input to the APV tool, which executes the workflow shown in Figure 5. In particular, the tool reads the information contained in the JSON file, determines the network structure (number of lines and buses, how they are connected, positions of sources and loads, how the current flows, etc.), and produces the complete UPPAAL model.

Concerning the UPPAAL model itself, the tool first generates the global declarations, including the necessary communication channels, global variables, and the `Isc`, `update` and `Reset` functions mentioned in Section III. Then, it creates the `FaultGenerator` TA and the CB TA. Finally, it produces the queries described in Section III-C, which are used to determine whether the protection system achieves selectivity or not. After the generation of the UPPAAL model is complete, the latter is automatically fed to the UPPAAL model checker, which checks one by one the defined properties. Finally, the results of the verification step are collected and stored in a separate file.

V. Tool Validation

We carried out various experiments with the APV tool, to evaluate its correctness and the feasibility of the whole approach. More precisely, we took two example networks from a technical paper describing selectivity provided by ABB [12], and we custom-built four more complicated networks expanding one of the first two examples, in order to test the scalability of the approach.

Taking examples from actual real-life applications and using them to test our tool ensures the correctness and functionality of the validation process. All examples satisfy the assumptions made by our tool, that is, they are all radial networks with a single source.

Table I shows the detailed information regarding the 10-bus custom-built example network, whose layout is shown in Figure 6. To test the scalability of the approach, we also built networks of 20, 30, 40, and 50 buses. In the table, the meaning of the columns is the following:

- `Iioc` is the current (in amperes) in the initial operating conditions.
- `Ith` is the threshold current (also in amperes).
- `Isc` is the current in the short circuit conditions (in kilo amperes).

The number shown in column `CB Type` of Table I identifies the settings used for the CB, where different type IDs correspond to different settings (hence, a CB of type 1 has different settings from one of type 2, for example).

TABLE I
DETAILED NETWORK INFORMATION AND CB SETTINGS FOR THE NETWORK OF FIGURE 6.

Line	Iioc [A]	Ith [A]	Isc [kA]	CB type
1	200	250	55	1
2	170	250	55	1
3	170	250	55	1
5	540	630	74	3
6	200	250	74	1
8	740	800	95	5
9	200	250	55	1
10	200	250	55	1
11	200	250	55	1
12	400	500	55	2
13	600	750	74	4

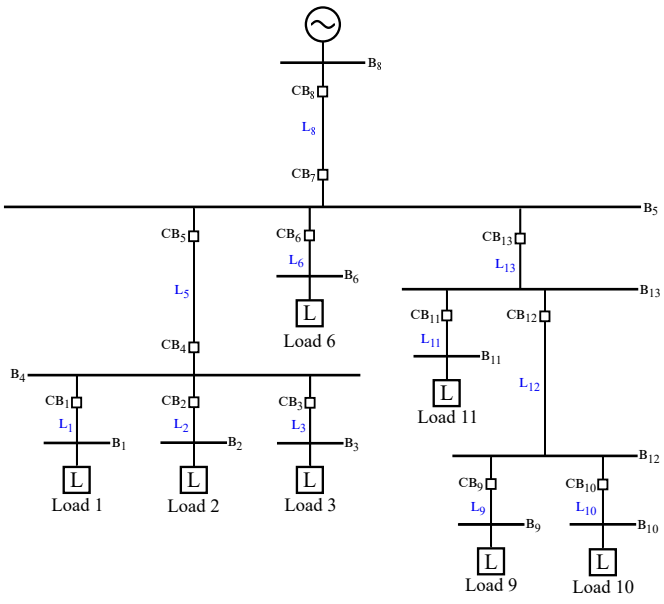

Fig. 6. Layout of the 10 bus custom-built network.

TABLE II
SUMMARY OF THE RESULTS OF THE VALIDATION EXPERIMENTS.

Prototype	no. of buses	no. of lines	comp. time [s]	no. of conditions checked
1	7	6	2.2	8
2	5	6	2.0	11
3	12	11	2.8	19
4	22	21	37.8	38
5	32	31	38.1	57
6	42	41	42.7	76
7	52	51	62.3	95

The validation of all 7 networks was successful: the APV prototype correctly created the complete UPPAAL model for all networks, verified all the queries and recorded the results. In addition, to validate the ability of the tool to recognize erroneous settings, for each network we also tested a case in which the settings for a randomly selected CB was changed in the JSON file to an incorrect one; in every case in which wrong settings were used, the tool correctly showed that selectivity was not achieved. In particular, the results produced by the tool were loaded back in UPPAAL, and a manual inspection of the outcome of the verification showed that selectivity was not achieved when there was a fault on the lines whose CB settings were changed. Inspecting the results through UPPAAL allows users to determine which CB opens prematurely, hence to focus on fixing the settings for the incorrectly configured CB and not for others.

Table II summarizes the results of the validation for all seven test protection systems. For each of the seven networks two tests were conducted: one using the correct CB settings and one in which the settings of a randomly-selected CB were changed. The results correctly showed that selectivity was achieved in all tests carried out with the correct settings, but it was not when the settings were altered.

The computation time, shown in Table II,[3] is the total time needed to read the JSON file, generate the UPPAAL model, run the verification process, and record the results. The time needed to carry out the two experiments (one with the correct settings, one with the wrong ones) for the same network was almost exactly the same, so we report only the value corresponding to the use of the correct settings. As the table shows, the time needed to carry out all necessary operations and checks for each network is in the order of the seconds—even for the 50-bus network, for which 95 conditions were checked. This is a very promising result that shows that the tool scales well for the kinds of networks targeted so far.

[3]All experiments were performed using UPPAAL 4.1.26-2 on a laptop equipped with Windows 11 and a 3.2 GHz AMD Ryzen 7 5800H processor and 16GB of RAM.

VI. CONCLUSION

In this paper, we presented a tool-supported approach for the automated verification of selectivity in LV distribution grids. The proposed methodology automatically generates the TA describing CBs' behavior, taking the relevant information from a JSON description file, and a set of queries that check whether selectivity is achieved. Then, it runs the models using UPPAAL, thus performing the formal verification of the stated properties. The approach was validated on a set of realistic networks. In this first implementation, we focused on single-sourced radial networks and on the time-current coordination mechanism. This study was the first step toward the implementation of an automated tool capable of formally verifying arbitrary networks.

REFERENCES

[1] P. Kundur and N. Balu, *Power System Stability and Control*, ser. EPRI power system engineering series. McGraw-Hill, 1994.

[2] T. Gonen, *Electric Power Distribution Engineering*. CRC Press, 2015.

[3] P. M. Anderson, C. Henville, R. Rifaat, B. Johnson, and S. Meliopoulos, *Power System Protection*, 2nd ed., ser. IEEE Press Series on Power and Energy Systems, G. K. Venayagamoorthy, Ed. John Wiley & Sons, Inc., Hoboken, New Jersey, 2022.

[4] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *Int. J. on Softw. Tools for Tech. Transf.*, vol. 1, no. 1-2, pp. 134–152, 1997.

[5] A. Trindade and L. Cordeiro, "Automated formal verification of stand-alone solar photovoltaic systems," *Solar Energy*, vol. 193, pp. 684–691, 2019.

[6] R. Cavada, A. Cimatti, S. Mover, M. Sessa, G. Cadavero, and G. Scaglione, "Analysis of Relay Interlocking Systems via SMT-based Model Checking of Switched Multi-Domain Kirchhoff Networks," in *2018 Formal Methods in Computer Aided Design (FMCAD)*, 2018, pp. 1–9.

[7] S. Ashraf, I. Evkay, O. Hasan, U. S. Selamogullari, and M. Baysal, "Formal Verification of Single Dual Setting Overcurrent Directional Relay Based Line Protection Logic for Smart Grids," in *2021 3rd Global Power, Energy and Communication Conference (GPECOM)*, 2021, pp. 227–232.

[8] S. M. N. R. Abadi, M. Mahmoodi, A. Fereidunian, G. Jahandoust, and H. Leasni, "Formal verification of fault location, isolation and service restoration in distribution automation using uppaal," in *2017 Conference on Electrical Power Distribution Networks Conference (EPDC)*, 2017, pp. 96–100.

[9] C. A. Furia, D. Mandrioli, A. Morzenti, and M. Rossi, *Modeling time in computing*. Springer Science & Business Media, 2012.

[10] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[11] "Low voltage circuit breakers, working with trip characteristic curves," White Paper, 2008.

[12] "Low voltage selectivity by ABB circuit breakers," White Paper, 2008.