Journal of Information and Organizational Sciences
Volume 49, Number 1 (2025)
Journal homepage: jios.foi.hr

JIOS

DOI: 10.31341/jios.49.1.3
Open Access

UDC 004.67Excel
Original Scientific Paper

Model Checking Access Control Protocol for Spreadsheets

Miro Zdilar[1*]

[1]Faculty of Organization and Informatics, University of Zagreb, Varaždin, Croatia
[*]Correspondence: miro.zdilar@gmail.com

PAPER INFO

Paper history:
Received 06 March 2025
Accepted 08 April 2025

Citation:
Zdilar, M. (2025). Model Checking Access Control Protocol for Spreadsheets. In Journal of Information and Organizational Sciences, vol. 49, no. 1, pp. 39-52

ABSTRACT

Spreadsheets are one of the most used software systems in business and academia. Since the first introduction of electronic spreadsheets for personal computers in 1979, spreadsheets have significantly evolved. With recent technological advancements and new features added, spreadsheets have become powerful computing platforms capable of complex analysis and modelling. However, numerous publications over the years described cases of spreadsheet errors. In focus of this research paper are spreadsheet errors caused by unauthorized access and modifications of spreadsheets in multi-user environments. Specifically, this paper is structured around formal verification of the novel ABAC4S (Attribute Based Access Control for Spreadsheets) protocol designed for prevention or detection of unauthorized modifications to spreadsheets in multi-user environments. We utilized a model checking approach to verify ABAC4S protocol rules for correctness.

Keywords: Spreadsheets, Spreadsheet Errors, Attribute Based Access Control Protocol, Unauthorized Spreadsheet Modifications, Model Checking

1. Introduction

Spreadsheets are widely used and can be considered as the most successful end-user programming systems. End-user programming systems allow end-users to build and execute powerful computer programs without the use of traditional programming languages and supporting development tools. It has been estimated that the number of end-user programmers outnumber traditional software programmers [1]. Spreadsheets are used in almost all companies in the US and Europe [2]. Modern enterprises use spreadsheets to support key processes such as capacity planning, financial reporting, stakeholder analysis, risk management, performance calculation, data transformation, cash-flows analysis, time-series transformations and simulations [3]. Despite their great success and importance, numerous publications over the years have described the importance of spreadsheet errors as well as the extent to which it has caused significant financial and reputational risks to individuals and organizations [4]. The European Spreadsheet Risk Interest Group (EuSpRIG), a non-profit and voluntary organization maintains a list of horror stories that illustrate problems with uncontrolled usage of spreadsheets [5].

In focus of this research paper are spreadsheet errors caused by unauthorized access and modifications of spreadsheets in multi-user environments. Specifically, this paper is structured around formal verification of the novel ABAC4S (Attribute Based Access Control for Spreadsheets) protocol designed for prevention or detection of unauthorized modifications to spreadsheets in multi-user environments. In [6], we have introduced Attribute Based Access Control conceptual model for spreadsheets. Herein we will provide thorough model checking of ABAC4S protocol and verify it for correctness. The remainder of this paper is organized as follows.

Section 2 provides a summary of related work in the field of automated detection of spreadsheet errors and controlled access for spreadsheet users in modern enterprises. In Section 3, the research methodology is presented structured around model checking of the proposed ABAC4S protocol. Afterwards, in Section 4, the ABAC4S protocol is presented with descriptions of model components and protocol rules. In Section 5, a brief introduction to model checking concepts is presented. In Section 6, formal verification of the proposed ABAC4S protocol with a symbolic model checker is provided. In Section 7, research results are discussed within the context of the overall spreadsheet research. Finally, in Section 8, the conclusions and reflections on research and verification procedures conducted are provided with proposals for future research opportunities.

2. Related Work

The tremendous success of spreadsheets and impact of spreadsheet errors triggered significant interest of the research community. In the following a critical review of the literature focused on taxonomy of spreadsheet errors, automated detection of spreadsheet errors, and controlled access for spreadsheet users in modern enterprises is provided.

2.1. Taxonomy of spreadsheet errors

Understanding types of spreadsheet errors is an important aspect of spreadsheet research and key to effective detection and prevention of spreadsheet errors.

Early studies listed types of errors detected without classification of spreadsheet errors. Brown and Gould [7] conducted reviews and experiments with volunteers experienced with spreadsheet use and development. As a part of the experiment, volunteers had to complete three tasks and create three different spreadsheets according to the instructions. Authors measured time required to complete the tasks, accuracy and visual appearance of final solution. An interesting part of this experiment was the use of a key logger [8] that recorded keystrokes of participants during the experiment and allowed insights to user behavior during completion of given tasks. Regardless of the limited number of participants, the experiment identified errors in formulae, mistyping, rounding and logical errors.

Galetta et al. [9] introduced two classes of spreadsheet errors. Authors distinguished between domain errors and device errors. The domain refers to the spreadsheet application area (e.g., accounting), while the device refers to the spreadsheet technology itself. For example, a mistake in logic due to a misunderstanding of depreciation is a domain error, but entering the wrong reference in the depreciation function SLN is a device error. Authors conducted an experiment with thirty accounting experts and thirty students to seek up to two errors introduced in each of six spreadsheets used during experiment. While accounting experts performed better in detection of domain errors, students demonstrated comparable performance in detection of device errors.

In one of the first attempts to offer a complete classification of errors, Panko and Halverson distinguished between quantitative and qualitative errors [10]. Quantitative errors are related to the current version of the spreadsheet, while qualitative errors refer to risky practices that might lead to an error in later stages of a spreadsheet's lifecycle. Panko and Halverson further divided quantitative errors into three subcategories: (i) mechanical errors, due to mistakes in typing or pointing, (ii) logic errors, due to choosing the wrong function or creating the wrong formula, and (iii) omission errors, due to misinterpreting the situation to be modeled. In critics to the above presented classification, Powell at al. [11] noted that this proposed classification does not take into account context of spreadsheet use and how each error was committed.

The taxonomy of errors developed by Rajalingham et al. [12] is one of the first attempts that introduced different spreadsheet user roles. This taxonomy is focused on user-generated errors and differentiates between developer and end-user errors. End-users are further classified as data inputter and interpreter. However, the given taxonomy classifies quantitative accidental errors as omission, alteration or duplication, without taking into consideration the possible errors caused by unauthorized changes in multi-user environments.

In recent years, researchers identified the need to relate types and occurrences of spreadsheet errors with the quality of the spreadsheets. Intuitively, a higher incidence of spreadsheet errors suggests that the overall quality of spreadsheet is low. O'Beirne presented an overview of information quality and data quality within the context of spreadsheets [13]. The author presented a comprehensive list of information quality attributes in the context of spreadsheet programs. In addition, the author presented checks and control procedures for spreadsheet information and quality processes.

Further refinement in spreadsheet quality research provided a set of domain specific metrics, used to measure concrete spreadsheet characteristic [14]. The presented quality model for spreadsheets is based on

the widely accepted ISO/IEC 9126 international standard for software product quality [15]. Authors provided a comprehensive analysis of ISO/IEC 9126 standard and mapped relevant quality attributes to spreadsheets.

2.2. Automated detection of spreadsheet errors

An automated method to infer data types from a spreadsheet was presented by Erwig and Burnett [16]. The proposed method for inferring types from spreadsheets is based on the concrete notion of units instead of the abstract concept of types. Authors used header information given by spreadsheets to derive units. In continuation of the presented concept around units, Ahamd et al. developed a type system for statically detecting spreadsheet errors [17]. The authors named the proposed model "unit checking" and presented a collection of rules that help identify weaknesses in spreadsheets that are likely to be errors. This model also relies on the concept of the header that defines common units for grouped cells. The working prototype based on the proposed model was developed for a specific version of Microsoft Excel spreadsheet application using the UCheck tool [18]. Authors validated performance of the UCheck tool in an experiment conducted with high school teachers [19]. Results of this experiment indicated that the tool effectively supports users in error correction.

High incidents of spreadsheet errors have led to a series of commercial software packages. Nixon and O'Hara provided structured assessments of several commercial auditing tools [20]. The test was designed to identify the success of software tools in detecting different types of errors, to identify how the software tools assist the auditor and to determine the usefulness of the tools. The assessment conducted by Nixon and O'Hara included the built-in auditing tool in Microsoft Excel spreadsheet [20]. Excel's built-in formula auditing tool supports visualization of spreadsheet formulas and error checking generated as result of formula evaluation.

In addition to research related to automated error detection, important to note is the work of Abraham and Erwig related to automation of spreadsheet testing [21]. The authors followed the original concept of mutation testing for general purpose programming languages and developed mutation operators for spreadsheets that allow generation of test cases.

Spreadsheets allow users to arrange data and metadata freely in a human readable format. To extract their content with automated tools, data practitioners need to perform manual inspections and data preparations. Mondrian system assists users with detection of multiregion layout templates in spreadsheets [36]. Mondrian comprises an automated approach to detect multiple data regions and an algorithm to compute layout similarity and identify templates with potential spreadsheet errors.

Recent spreadsheet research is focused on the application of large language models to improve spreadsheets quality. A team of researchers from Microsoft Corporation developed the FLAME language model for spreadsheet formulae [22]. FLAME uses the Microsoft Excel specific formula tokenizer and other techniques to achieve competitive performance with a substantially smaller model (60 million parameters) and training dataset, compared to other large language models such as Codex. Researchers used a training dataset of 972 million formulas extracted from a corpus of 1,8 million Excel workbooks. FLAME was evaluated on three different tasks for Excel formulas: last-mile repair, autocompletion and syntax reconstruction. The presented FLAME language model outperformed larger language models, such as Codex-Davinci (175 billion parameters), Codex-Cushman (12 billion parameters), and CodeT5 (220 million parameters), in 6 out of 10 experimental settings [22].

2.3. Access control for spreadsheets

Access control and authorization are key components of information technology systems in multi-user environments. Korman et al. evaluated existing access control models in the context of different business scenarios [23]. They also provided a unified metamodel capable of expressing access policies for all evaluated models. In [6] we have conducted an evaluation of common access control models, as well as their suitability for spreadsheet use in multi-user environments. Below is a summary of ABAC (Attribute Based Access Control) model advantages for spreadsheet applications, while detailed evaluation is provided in [6]:

- The ABAC model is based on dynamic attributes, where object attributes fit to the proposed model of spreadsheet resources and corresponding attributes.
- A hierarchy of spreadsheet resources can be modelled with ABAC conditions and access rules determinations. This property prevents conflicts in access resolutions and simplifies prototype implementation.
- Deployment opportunities for ABAC with spreadsheets are flexible and allow early prototype implementation as a corrective access control system. This minimizes impact on users and generally accepted spreadsheet user interface.

- Complexity of the ABAC model for spreadsheets depends on the number of spreadsheet resource attributes.
- The dynamic nature of modern cloud-powered spreadsheets and extensions to spreadsheet formula language fits nicely to ABAC's dynamic attribute concept. Potential new functionalities and modules added in cloud-powered spreadsheet can be integrated within existing ABAC concepts.

The National Institute of Standards and Technology (NIST) published a guideline with definitions for the ABAC model [24]. The guideline provides definitions and considerations for using ABAC to improve information sharing and design of systems, while maintaining control of that information. The concepts and terminology for ABAC presented in that document have been instrumental for the design and verification of ABAC4S protocol presented in this paper.

3. Research Methodology

The research methodology in this paper follows the Design Science Research approach [35]. We started our research journey with comprehensive literature review to understand existing knowledge base. We followed a focused approach on work related to taxonomy of spreadsheet errors and automated methods applied to detection and prevention of spreadsheet errors as described in Section 2.

Based on the existing knowledge base and literature review conducted, we identified research opportunities to address unauthorized spreadsheet changes and errors in multi-user environments. Aligned with the Design Science Research approach and requirements to control user interaction with spreadsheets in multi-user environments, we formulated our first research goal:

- RG1 – Develop formal description of access control protocol capable of controlling user's interaction with spreadsheets in multi-user environments.

In search of the appropriate access control protocol, we further investigated the concept of spreadsheets represented as collection of resources [6]. The first research goal is the basis for the second research goal, in that it provides specification of the access control protocol that can be formally verified. The next step in our research is to evaluate access control properties with a model checking tool, which leads to our second research goal:

- RG2 – Evaluate correctness property of the proposed access control protocol with model checking approach.

To address the second research goal further, we used NuSMV model checker to verify correctness property of the proposed access control protocol [25]. The selection of the NuSMV model checker has been primarily driven by the richness of supported SMV language and its capability to specify hierarchical SMV modules that correspond to the natural hierarchy of spreadsheet resources.

Even though presented research methodology is structured around two formulated research goals, we followed iterative research through experimentation and simulation to refine outcomes of the conducted research. Initial results of the model verification and provided counterexamples were instrumental for access protocol improvements and protocol rules redesign. Appendices, if included, follow the main text. Each appendix should be lettered, e.g., "Appendix A". To properly format appendix title, use Heading Unnumbered style from the styles menu.

4. ABAC4S

ABAC4S protocol is designed to control unauthorized activities on spreadsheets in multi-user environments. The core idea of the proposed protocol revolves around spreadsheet representation as a collection of resources [6]. In modern cloud-based spreadsheets, resources are building blocks manipulated with a native spreadsheet formula language or custom computational modules constructed with external programming languages. Spreadsheet resources and their attributes are bound by ABAC4S rules and allow granular control of resource states during a spreadsheet's lifecycle. The ABAC4S protocol specification consists of five distinct parts [27]:

1. The *Service* to be provided by the protocol
2. The *Assumptions* about the environment in which the protocol is executed
3. The *Vocabulary* of data flows used to implement the protocol
4. The *Encoding* (format) of each data flows in the vocabulary
5. The *Procedure rules* guarding the consistency of data flows and correctness of *Service* to be provided by the protocol

4.1. ABAC4S Protocol Service Specification

The ABAC4S protocol is defined on the conceptual level of modern cloud-based spreadsheets and is agnostic form specific commercial implementations of spreadsheets. In addition, ABAC4S protocol specifications provided in this paper are based on set-oriented data structures and data flows suitable for translation to model checking tools and verification of correctness for protocol rules.

4.2. Assumptions about ABAC4S Protocol Environment

The environment in which the protocol is executed consists of the ABAC4S conceptual model and four generic user roles typically found in multi-user environments: developer, tester, analyst and manager. The ABAC4S protocol is not restricted to only 4 specified users and can be easily extended to unlimited number of user roles depending on specific deployment needs. Four generic user roles are selected to limit the complexity of the model and prevent state space explosion during model checking. The ABAC4S protocol definition in this paper focuses on accurate and complete specification of data flows and procedure rules, while implementation for protocol execution on commercial spreadsheets is left for specific deployment scenarios.

4.3. ABAC4S Protocol Vocabulary

The spreadsheet conceptual model is visually presented in Figures 1 and 2, followed with the model of ABAC4S access rules for spreadsheets presented in Figure 3. Detailed description of spreadsheet conceptual model is provided in original paper [6].

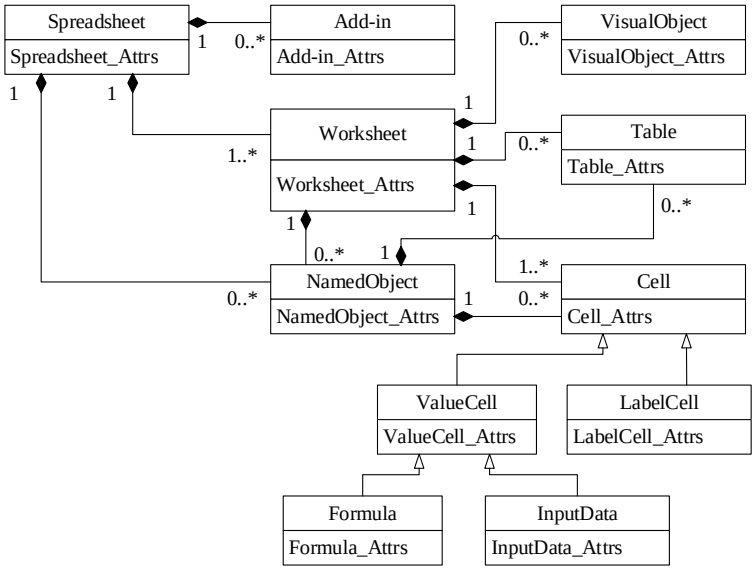

Figure 1. Metamodel of spreadsheet resources and associated attributes [6].

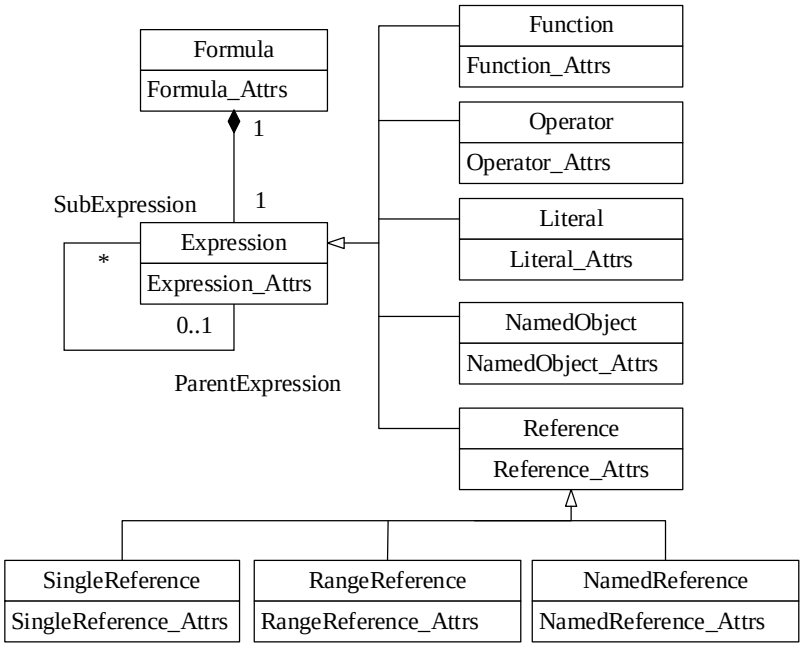

Figure 2. Spreadsheet formula metamodel [6].

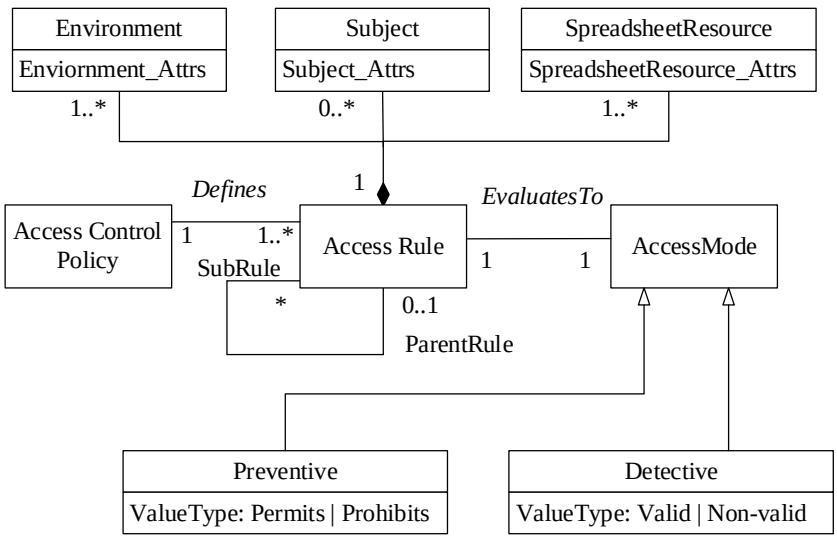

Figure 3. ABAC4S access rules for spreadsheets [6].

Spreadsheet stages during lifecycle phases are modelled with finite sequence of state transitions as follows:

$$S_0 \xrightarrow{\Delta S_0(MU)} S_1 \xrightarrow{\Delta S_1(MU)} S_2 \xrightarrow{\Delta S_2(MU)}, \dots, S_e. \quad (1)$$

where S_0 is the initial state of the spreadsheet ("first creation"), S_e is the final state of spreadsheet ("end of lifecycle"), $\Delta S_j(MU)$ are transitions between spreadsheet states caused by modifications M of user U on spreadsheet resources SR.

$$U \in [developer, tester, analyst, manager]. \quad (2)$$

Modifications M are determined by comparing affected spreadsheet resource at states S_{j+1} and S_j. Transitions $\Delta S_j(MU)$ are modeled as triplets with the following structure:

$$\Delta S_j(MU) = \left(U, M, SR_j\right). \quad (3)$$

In the proposed ABAC4S protocol vocabulary, access rules are modelled as quadruples with the following structure:

$$(U, A, SR, E). \quad (4)$$

A is a set of actions that user might perform on spreadsheet resource represented with following enumerated list:

$$A \in [CREATE, READ, UPDATE, DELETE]. \quad (5)$$

These actions are usually denoted with the CRUD acronym. The proposed ABAC4S protocol is not limited to four CRUD actions, and if needed in specific deployment scenarios, the number of actions could be reduced or extended.

SR represents a set of spreadsheet resources and corresponding resource attributes on which user U can perform action A.

E are dynamic environmental conditions, independent of the users and the spreadsheet resources that may be used as attributes at decision time to influence an access decision. Examples of environmental conditions include time, location, threat level, or temperature [6].

4.4. ABAC4S Protocol Encoding

For the ABAC4S protocol definition presented herein, we denoted spreadsheet transitions and access rules as abstract set-oriented data structures. This will allow us to express procedure rules with a specific order of evaluation between sets and model hierarchies between spreadsheet resources with set compositions. Abstract data structures presented in the protocol definition can be transformed into programming language data structures or encoded to other formats like XML (eXtensible Markup Language) or JSON (JavaScript Object Notation) messages during specific implementation scenarios.

4.5. ABAC4S Protocol Rules

The procedure rules for the ABAC4S protocol are defined as follows:

Rule 1 – Priority of Actions
Actions assigned to users are evaluated in the following order:

$$DELETE > CREATE > UPDATE > READ. \quad (6)$$

Delete action has the highest priority. For example, if the access rule permits the user to delete a specific spreadsheet resource, the user is also allowed to create, update, and read the corresponding spreadsheet resource.

Rule 2 – Access Rule Inheritance
All spreadsheet's resources inherit access rules applicable to their parents.

Rule 3 – Spreadsheet Valid State
Spreadsheet is in valid state S_{j+1}, iff Rule 1 (Priority of actions) and Rule 2 (Access Rule Inheritance) are satisfied for all affected spreadsheet resources during spreadsheet state transition from S_j to S_{j+1}.

5. Model Checking

Model checking is a model-based verification procedure designed to automatically verify properties of finite state systems [26], [28]. The core principle behind a model checking procedure is exhaustive exploration of states to verify whether a given system model satisfies certain properties.

Transition state machines are used in model checking to represent the behavior of the system. A common method for representing transition state machines are Kripke structures. A Kripke structure M is represented as an ordered sequence of four objects:

$$M = (S, I, R, L). \tag{7}$$

S: finite set of states
I: set of initial states $I \subseteq S$
R: transition relation $R \subseteq S \times S$
L: interpretation function $L: S \rightarrow 2^{AP}$

For each state $s \subseteq S$ there is a possible successor state $s' \subseteq S$ specified with transition relation R. The interpretation function L labels each state with Atomic Propositions (AP) which are Boolean variables and the evaluations of expressions in that state [26]. A finite path π from some state $s \epsilon S$ is a sequence of states $\pi = s_0, s_1, \ldots, s_n$ such that $s_0 = s$ and $R(s_i, s_{i+1})$ holds for all $0 \leq i < n$ [26].

Emerson and Clarke introduced model checking [29] and Computational Tree Logic (CTL) as a combination of linear temporal logic and branching-time logic [30]. In model checking, temporal logic is used to express system specifications (properties) denoted as ϕ. CTL combines path quantifiers and temporal operators to describe events associated with single computation path.

CTL path quantifiers are as follows:

- ***A*** – for **A**ll paths from a certain state on
- ***E*** – there **E**xists at least one single path from a certain state

CTL temporal operators are as follows:

- ***X*** ϕ – ϕ holds ne**X**t time
- ***F*** ϕ – ϕ holds sometime in the **F**uture
- ***G*** ϕ – ϕ holds **G**lobally in the future
- p ***U*** ϕ – ϕ holds **U**ntil ϕ holds

CTL allows modeling complex behavior of the systems, where temporal operator must always be preceded by a path quantifier. Figure 4, adapted from [31] visually represents the meaning of CTL path and temporal operators.

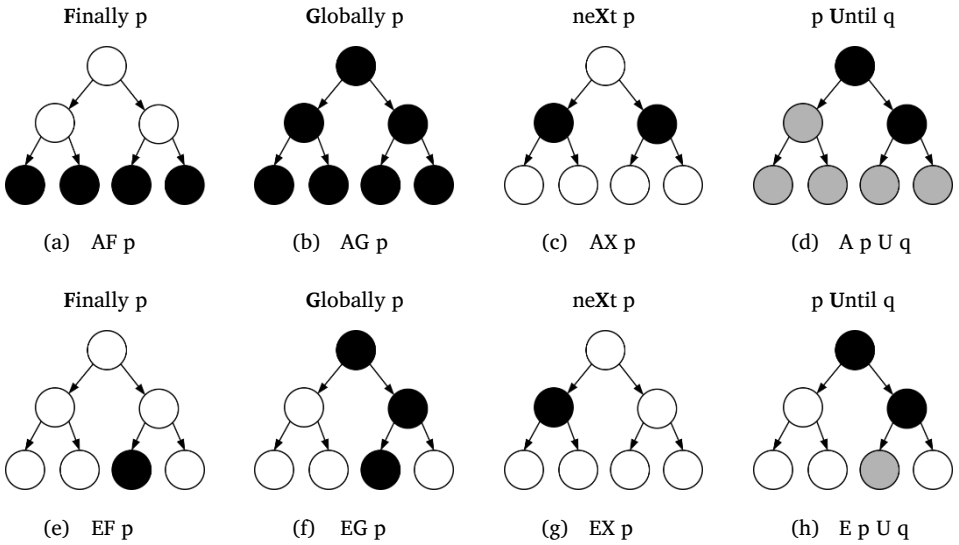

Figure 4. CTL path and temporal operators [31].

In practical model checking applications, system model M is described semantically with a Kripke structure and the specifications (properties) are described with formulae ϕ in the applicable form of temporal logic. The decision procedure conducted by a model checker tool decides whether $M \vDash \phi$. Operator $\vDash$ meaning is "specification ϕ is satisfied by structure M".

6. Model Checking the ABAC4S Protocol

We performed model checking of the proposed ABAC4S protocol for spreadsheets with the NuSMV symbolic model checker [25]. Original SMV model checking tool has been developed at the Carnegie Mellon University [32]. NuSMV is a modern variant of original SMV symbolic model checker with compatible SMV language syntax and advanced architecture that allows textual construction of hierarchical models and verification of very large number of states [33].

The system model is a transition system with a set of states and transition relations that specifies the behavior of the system. In SMV language, a system is defined as a module, beginning with the keyword `MODULE`. The module consists of an encapsulated collection of declarations (such as `VAR`, `INIT`, `ASSIGN`, etc.) that depend on the nature of the analyzed problem and specific parameters. A module's state variables declaration begins with the keyword `VAR`. In general, model checker tools are limited to only few data types and the SMV language allows for Boolean values, enumeration of constants, or other modules for constructing hierarchical models. The set of initial states can be specified with simple logical statements or conjunctions of equations associated with the initial state of the system. The transition relation of a module starts with the keyword `ASSIGN` and may be limited to single statement or complex set of equations. An assignment statement is structured as the next step evaluation, where the right-hand side allows the construction of complex expressions built with Boolean operators, integer arithmetic and case constructs with conditions.

The main challenge during the modeling of the spreadsheet conceptual model with the SMV language has been the abstraction of the provided model (Figures 1 and 2) with suitable SMV constructs. We represented each spreadsheet resource with a corresponding SMV module. The hierarchy of SMV modules follows the natural hierarchy of spreadsheet resources defined in the spreadsheet conceptual model (Figures 1 and 2).

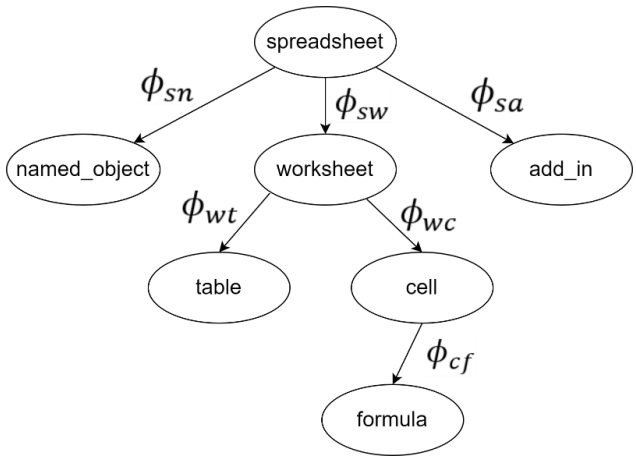

Figure 5. Hierarchy of spreadsheet resources as SMV language modules.

In addition, we wanted to explore all possible access role assignments and evaluate ABAC4S protocol for all combinations of four defined user roles and CRUD actions. In such a scenario, correct protocol behavior should detect potential conflicts and defined priority of actions and access inheritance rules should ensure correct resolution of detected conflicts in the consecutive model state. Below is the hierarchical model of spreadsheet resources specified in SMV.

```
MODULE spreadsheet_t()
VAR
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};
    add_in:add_in_t();
    named_object:named_object_t();
    worksheet:worksheet_t();

MODULE add_in_t()
VAR
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};

MODULE named_object_t()
VAR
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};

MODULE worksheet_t()
VAR
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};
    table:table_t();
    cell:cell_t();

MODULE table_t()
VAR
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};

MODULE cell_t()
VAR
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};
    formula:formula_t();

MODULE formula_t()
VAR
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};

MODULE main
VAR
    spreadsheet:spreadsheet_t();
```

As listed in the above specifications, we utilized the capability of SMV language and build hierarchical modules that correspond to the natural hierarchy of spreadsheet resources. The above modules represent all possible state transitions $\Delta S_j(MU)$ for each spreadsheet resource, assigned users and CRUD actions. To prevent state space explosion, we abstracted and simplified each spreadsheet resource to a bare minimum. For example, an additional two attributes on module `spreadsheet_t()` can be added with an enumerated list of constants, and a next case assignment for added attributes sharing the same structure with simplified model specification.

```
MODULE spreadsheet_t()
VAR
    attributes:{spreadsheet_attribute1,spreadsheet_attribute2};
    role:{developer,tester,manager};
    a:{create,read,update,delete};
    add_in:add_in_t();
    named_object:named_object_t();
    worksheet:worksheet_t();
```

Transitions to new states are modeled in SMV with next-case statements within the `ASSIGN` language construct. ABAC4S protocol rules for priority of actions and access rule inheritance are specified with a

complex conjunction statement from relevant spreadsheet resource properties. As visually represented in Figure 5, there are six conjunction statements ($\phi_{sn}, \phi_{sw}, \phi_{sa}, \phi_{wt}, \phi_{wc}, \phi_{cf}$) that correspond with the hierarchical representation of spreadsheet resources. In order to correctly specify both protocol rules for priority of actions and access rule inheritance, the correct transition to the next state for the hierarchically lowest spreadsheet resource (formula) should be evaluated as a composition of all statements on the path to the root spreadsheet resource ($\phi_{cf}, \phi_{wc}, \phi_{sw}$). Due to complexity of specifications and limited space in this publication, a fragment of SMV code for ϕ_{sw} next-case conjunction statement that specify logic for priority of actions and access inheritance protocol rules is listed below. The complete SMV protocol specification can be fetched from author's GitHub repository at [34].

```
next(spreadsheet.worksheet.a) :=
  case
    spreadsheet.role=spreadsheet.worksheet.role) & \
    (spreadsheet.a=read) & (spreadsheet.worksheet.a in \
    update,create,delete}): read;

    (spreadsheet.role=spreadsheet.worksheet.role) & \
    (spreadsheet.a=update) & (spreadsheet.worksheet.a in \
    read,create,delete}): update;

    (spreadsheet.role=spreadsheet.worksheet.role) & \
    (spreadsheet.a=delete) & (spreadsheet.worksheet.a in \
    read,create,update}): delete;

    (spreadsheet.role=spreadsheet.worksheet.role) & \
    (spreadsheet.a=create) & (spreadsheet.worksheet.a in \
    read,update,delete}): create;

    TRUE : spreadsheet.worksheet.a;
  esac;
```

After finalization of model construction and formal specification of the spreadsheet conceptual model and ABAC4S protocol rules, we conducted model checking with the NuSMV model checker. We utilized NuSMV in interactive mode, and executed CTL temporal logic property checks in NuSMV built-in shell.

CTL temporal logic specification with the following structure has been used to verify correctness of protocol rules for priority of actions and access inheritance:

$$AG\ (p\ \rightarrow AF\ q). \tag{8}$$

The CTL temporal logic specification above should be interpreted as "for all execution paths globally, when condition p occurs it is always followed by condition q". If we apply the above generic CTL specification i.e. for the table spreadsheet resource, the specific CTL syntax is as follows:

```
check_ctlspec -p "AG \
((spreadsheet.role=spreadsheet.worksheet.role) & \
(spreadsheet.worksheet.role= \ spreadsheet.worksheet.table.role) & \
(spreadsheet.a=read) & (spreadsheet.worksheet.a=read) & \
(spreadsheet.worksheet.table.a in {update,create,delete}) \
-> AF spreadsheet.worksheet.table.a=read)"
```

As a result of the above CTL temporal logic specification check, the NuSMV model checker confirms that the above specification is satisfied by given model:

```
NuSMV >
-- specification AG (((((spreadsheet.role = spreadsheet.worksheet.role &
spreadsheet.worksheet.role = spreadsheet.worksheet.table.role) & spreadsheet.a =
read) & spreadsheet.worksheet.a = read) & spreadsheet.worksheet.table.a in
(update union create) union delete) -> AF spreadsheet.worksheet.table.a = read)
is true
NuSMV >
```

NuSMV model checker evaluates the above CTL specification to true, thus formally verifying correct conflict resolution and correct behavior of two protocol rules in case of a table spreadsheet resource. Appropriate CTL specifications for other spreadsheet resources follow the same generic structure, however conjunction statements are growing in complexity for hierarchically lower spreadsheet resources due to longer evaluation path to the root spreadsheet resource.

7. Discussion

Spreadsheet related research is a rich knowledge base with great scientific contributions. Many authors addressed the need to control spreadsheet errors as described in Section 2.2. Results of our research focused on unauthorized spreadsheet changes and errors in multi-user environments could usefully be combined with unit errors detection in spreadsheet [18], other commercial spreadsheet auditing tools [20] and modern large language models to improve spreadsheet quality [22]. Additional strength of the proposed ABAC4S protocol is result of our user-centric approach followed during ABAC4S protocol development and specification. This approach permits organizations to retain investment in their spreadsheets. The negative side of our approach is the requirement to document organizational dynamics and user roles in the format suitable for ABAC4S protocol.

To address our first research goal, we developed the novel ABAC4S protocol specifications based on spreadsheet representation as collection of resources. Defined protocol addresses the need identified to control user's interaction with spreadsheets on granular level of spreadsheet resources. We utilized set-oriented data structures and data flows for modeling transitions between spreadsheet states.

To address our second research goal, we converted defined ABAC4S protocol specifications to SMV language and conducted model checking to verify correctness of the protocol rules. During model construction, we used abstraction and refinement of model characteristics to reduce model complexity and prevent state space explosion during verification with model checking tools. We designed modules to represent all possible state transitions for each spreadsheet resource with associated users and actions as described in Section 4.3. Thus, we ensured that the model covers all possible realistic scenarios where users in multi-user environments might have roles of various complexities. We overcome various challenges with model abstraction and state space explosion associated with the model checking tools. Even though we reduced the number of allowed actions and number of user roles to four, the model checking tool must explore 16^7 (more than 268 million) possible states. To illustrate the importance of appropriate model abstraction and its impact to the state space explosion, if we increased the number of modeled actions and user roles to five, possible explorable state would grow to 25^7. This small increase in the model complexity resulted in a more than 22 times larger model state space.

8. Conclusion

In this study, we presented results of research structured around model checking of novel ABAC4S protocol for spreadsheets. To our best knowledge, application of the model checking technique in verification of spreadsheet related research problems brings new perspective in spreadsheet research. We provided modeling guidelines and insights into how to convert ABAC4S protocol specifications to the language accepted by the model checking tool. Model checking confirms correctness of the defined ABAC4S protocol rules and appropriate resolution in case a conflict of access rights is detected. We will explore opportunities to further automate generation of machine-readable user's roles and implement developed ABAC4S protocol in various multi-user environments.

References

[1] C. Scaffidi, M.Shaw, and B. A. Myers, "Estimating the numbers of end users and end users programmers", In Proc. of VL/HCC '05, pp. 207-214, 2005.

[2] L. Bradley and K. McDaid, "Using Bayesian Statistical Methods to Determine the Level of Error in Large Spreadsheets", Proceedings of the International Conference on Software Engineering, pp. 351–354., 2009.

[3] T. Reschenhofer and F. Matthes, "A Framework for the Identification of Spreadsheet Usage Patterns", Proceedings of the European Conference on Information Systems, 2015.

[4] K. Rajalingham, D. Chadwick, B. Knight, and D. Edwards, “Quality Control in Spreadsheets: A Software Engineering-Based Approach to Spreadsheet Development,” Proc. 33rd Hawaii Int’l Conf. System Sciences, pp. 1-9, 2000.

[5] P. O’Beirne, F. Hermans, T. Cheng, M. P. Campbell, European Spreadsheet Risk Interest Group, “https://eusprig.org/research-info/horror-stories/”, [Accessed: Feb. 23, 2025].

[6] M. Zdilar, “Attribute Based Access Control Metamodel for Spreadsheet Programs”, 35th International Scientific Conference CECIIS 2024. Varaždin: University of Zagreb, Faculty of Organization and Informatics, pp. 409-416., 2024.

[7] P. Brown, J. Gould, “An experimental study of people creating spreadsheets”, ACM Transactions on Office Information Systems 5, pp.258–272, 1987.

[8] W. J. Doherty, W. Pope, “Computing as a tool for human augmentation”, IBM Tech Rep. RC-11622, 1986.

[9] F. Galletta, D. Abraham, M. El Louadi, W. Leske, Y. Pollalis and J. Sampler, “An empirical study of spreadsheet error-finding performance”, Accounting, Management & Information Technology Vol. 3 No. 2, pp. 79–95, 1993.

[10] R. Panko and R. Halverson, “Spreadsheets on trial: a survey of research on spreadsheet risks”, Proceedings of the 29th Annual Hawaii International Conference on Systems Sciences, pp. 326–335, 1996.

[11] S. G. Powell, K. R. Baker and B. Lawson, “A critical review of the literature on spreadsheet errors”, Decision Support Systems, pp. 128-138, 2008.

[12] K. Rajalingham, D. Chadwick, B. Knight, “Classification of spreadsheet errors”, Proceedings of the European Spreadsheet Risks Interest Group Annual Conference, Greenwich, England, pp. 23-34, 2000.

[13] P. O’Beirne, “In Pursuit of Spreadsheet Excellence”, Proceedings of EuSpRIG, pp. 171-185, 2008.

[14] J. Cunha, J. Fernandes, C. Peixoto and J. Saraiva, “A Quality model for Spreadsheets”, Proceedings of the 8th International Conference on the Quality of Information and Communications Technology, pp. 231-236, 2012.

[15] ISO(2001), “ISO/IEC 9126-1: Software engineering-product quality-part 1: Quality model,” Geneva, Switzerland, 2001.

[16] M. Erwig and M. M. Burnett, “Adding apples and oranges”, In Proc. Of PADL '02, pp. 173-191, 2002.

[17] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi, “A type system for statically detecting spreadsheet errors”, In Proc. of ASE '03, pp. 174-183, 2003.

[18] R. Abraham and M. Erwig. “Ucheck: A spreadsheet type checker for end users. Journal of Visual Languages and Computing”, Vol. 18, pp. 71-95, 2007.

[19] R. Abraham, M. Erwig, and S. Andrew, “A type system based on enduser vocabulary”, In Proc. of VL/HCC, pp. 215-222, 2007.

[20] D. Nixon, M. O’Hara, “Spreadsheet Auditing Software”, In Proc. Of EuSpRIG, 2000.

[21] R. Abraham and M. Erwig, “Mutation Operators for Spreadsheets”, IEEE Transactions on Software Engineering”, Vol. 35 No. 10, 2009.

[22] H. Joshi, A. Ebenezer, J. Cambronero, S. Gulwani, A. Kanade, V. Lee, ... & G. Verbruggen, “FLAME: A small language model for spreadsheet formulas”, arXiv preprint arXiv:2301.13779, 2023.

[23] M. Korman, R. Lagerström, M. Ekstedt, “Modeling enterprise authorization: a unified metamodel and initial validation.” Complex Systems Informatics and Modeling Quarterly, (7), 1-24, 2016.

[24] C. T. Hu, “Attribute based access control (ABAC) definition and considerations.”, NIST, 2014.

[25] A. Cimati, E. Clarke, F. Giunchiglia, M. Roveri, “NuSMV: A new symbolic model verifier”, Computer Aided Verification: 11th International Conference, CAV’99 Trento, Italy, July 6–10, 1999 Proceedings 11 (pp. 495-499). Springer Berlin Heidelberg, 1999.

[26] E. M. Clarke, O. Grumberg, D. Peled, “Model Checking”, MIT Press, 2000.

[27] J. G. Hoizmann, “Design and Validation of Computer Protocols”, Prentice Hall, 1991.

[28] C. Baier, J. P. Katoen, “Principles of Model Checking”, MIT Press, 2008.

[29] E. M. Clarke, E. A. Emerson, “Design and synthesis of synchronization skeletons using branching time temporal logic”, Workshop on Logic of Programs, ser. Lecture Notes in Computer Science, vol. 131, pp. 52–71., 1981.

[30] E. A. Emerson, E. M. Clarke, “Characterizing correctness properties of parallel programs using fixpoints,” In Proceedings of the 7th Colloquium on Automata, Languages and Programming, pp. 169–181., 1980.

[31] T. Reinbacher, “Model checking and static analysis of Intel MCS-51 Assembly Code”, Wien, 2012.

[32] K. McMillan, “Symbolic Model Checking”, Kluwer Academic Publishers, 1993.

[33] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen, “Systems and software verification: model-checking techniques and tools”, Springer Science & Business Media, 2013.

[34] M. Zdilar, “https://github.com/mirogit/abac-spreadsheets”, GitHub repository, 2025.

[35] A. R. Hevner, S.T. March, J. Park, S. Ram, “Design Science-Design Science in Information Systems Research”, MIS Quarterly 28, pp.75-105, 2004.

[36] G. Vitagliano, L. Reisener, L. Jiang, M. Hameed, F. Neumann, “Mondrian: Spreadsheet layout detection”, In Proceedings of the 2022 International Conference on Management of Data, pp. 2361-2364., 2022.