University of Zagreb

FACULTY OF ORGANIZATION AND INFORMATICS

Miro Zdilar

AUTOMATED VERIFICATION OF SPREADSHEET PROGRAMS

DOCTORAL THESIS

Varaždin, 2025.

Sveučilište u Zagrebu

FAKULTET ORGANIZACIJE I INFORMATIKE

Miro Zdilar

AUTOMATSKA VERIFIKACIJA TABLIČNIH KALKULATORA

DOKTORSKI RAD

Varaždin, 2025.

DOCTORAL THESIS INFORMATION

I. AUTHOR

Name and surname	**Miro Zdilar**
Place and date of birth	**Pula, June 24th 1970**
Faculty name and graduation date	
Current employment	**Raiffeisenbank Austria dd, Zagreb**

II. DOCTORAL THESIS

Title	**Automated Verification of Spreadsheet Programs**
Number of pages, figures, tables, appendices, bibliographic information	**Pula, June 24th 1970**
Scientific area and field in which the title has been awarded	**Social Sciences, Information and Communication Sciences**
Supervisor	**Prof. Markus Schatten, PhD**
Faculty where the thesis was defended	**University of Zagreb, Faculty of Organization and Informatics**
Mark and ordinal number	

II. GRADE AND DEFENSE

Date of doctoral thesis topic acceptance	**April 9th 2024**
Date of doctoral thesis submission	
Date of doctoral thesis positive grade	
Grading committee members	
Date of doctoral thesis defense	
Defense committee members	
Date of promotion	

University of Zagreb

FACULTY OF ORGANIZATION AND INFORMATICS

Miro Zdilar

AUTOMATED VERIFICATION OF SPREADSHEET PROGRAMS

DOCTORAL THESIS

Mentor: Prof. Markus Schatten, PhD

Varaždin, 2025

University of Zagreb

FAKULTET ORGANIZACIJE I INFORMATIKE

Miro Zdilar

AUTOMATSKA VERIFIKACIJA TABLIČNIH KALKULATORA

DOKTORSKI RAD

Mentor: Prof. Markus Schatten, PhD

Varaždin, 2025

to my wife, Melita
and our angels Dominik and Gita,
With love

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to all those who supported my research and made this thesis possible. At first and foremost, my supervisor, Professor Markus Schatten PhD, played a pivotal role in conducting my thesis research. Thank you, Professor Schatten!

Special thanks go to my colleagues and friends for evaluating developed prototypes during my research and all the energy and effort they put into explanation of their laboratory processes and analytical measurements.

To all my friends for their encouragement to stay focused when I was exhausted and who never ran away during my stories about spreadsheet development and verification.

I am eternally grateful for the support of my friend Miljenko Košiček, whose memory will forever motivate my work.

I would also like to thank my parents, who believed in me and encouraged me throughout my graduate and postgraduate studies.

Finally, I am deeply grateful to my happy trio, especially to my wife Melita, for her invaluable support, encouragement and understanding throughout this challenging research journey. Without their patience and support, this thesis would never have been completed.

Abstract in English

Spreadsheets are widely used and can be considered as one of 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. The combination of a visual grid, powerful built-in functions, graphing capabilities, and user-friendliness makes spreadsheets an indispensable tool for anyone needing to work with structured data. In enterprise environments, spreadsheets are deeply integrated into core business operations. Their use extends beyond simple data entry to underpin various business processes, particularly where data analysis, forecasting, and decision-making are paramount. With recent technological advancements and new features added, spreadsheets have become powerful computing platforms capable of complex analysis and modelling. The spreadsheet research consistently demonstrates that spreadsheet errors are omnipresent, with studies revealing error rates in a substantial percentage of models, ranging from simple data entry mistakes to complex logical flaws in formulae. While there isn't one universally adopted taxonomy of spreadsheet errors that supersedes all others, core distinction between quantitative and qualitative errors remains central idea for research. Quantitative errors directly lead to incorrect numerical values or logical flaws in the spreadsheet program direct results. Qualitative errors do not immediately produce incorrect numerical values but represent poor design practices, incorrect assumptions, or structural flaws that degrade the spreadsheet's quality during the lifetime of spreadsheet. Qualitative errors increase the likelihood of future quantitative errors, or make the spreadsheet difficult to understand, maintain, or debug. The impacts of such errors can be severe, leading to flawed financial forecasts, incorrect scientific analyses, misguided business decisions, legal liabilities, and even corporate collapses.

To address challenges associated with spreadsheet errors, research in spreadsheet quality has focused on methods for finding (detection) and avoiding (prevention) spreadsheet errors. Detection methods range from manual auditing and peer review to more sophisticated techniques involving code analysis tools, testing-based techniques and data visualization for anomaly detection. Prevention strategies emphasize best practices in spreadsheet design including modularity, consistency with formatting and naming conventions and refactoring of spreadsheet formulae with modern structured references and named objects.

In focus of this thesis is automated quality assurance for spreadsheets in multi-user environments. Specifically, this thesis is structured around the novel ABAC4S (Attribute Based Access Control for Spreadsheets) protocol designed for automated quality assurance of

spreadsheets in multi-user environments. Novel ABAC4S protocol for automated quality assurance of spreadsheet programs uniquely addresses both methods in focus of quality assurance research, finding (detection) and avoiding (prevention) of spreadsheet errors and quality issues. As part of the research presented in this thesis, correctness of presented ABAC4S protocol has been formally verified with model checking approach.

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

Abstract in Croatian

Tablični kalkulatori se koriste u raznim domenama ljudske djelatnosti i mogu se smatrati jednim od najuspješnijih programskih sustava za krajnje korisnike. Programski sustavi za krajnje korisnike omogućuju im da grade i izvršavaju složene računalne programe bez upotrebe tradicionalnih programskih jezika i pratećih razvojnih alata. Tablični kalkulatori objedinjuju ugrađene funkcije za izvođenje kompleksnih izračuna i alate za vizualizaciju podataka. Jednostavnost korištenja, intuitivno sučelje i mogućnost brzog rješavanja složenih programskih zadataka čine tablične kalkulatore moćnim alatom za obradu podataka.

U poslovnim okruženjima, tablični kalkulatori su nezamjenjiv alat za podršku poslovnim procesima. Posebno je značajna uloga tabličnih kalkulatora u procesima predviđanja i donošenja poslovnih odluka na temelju velike količine raznorodnih podataka. Pojavom modernih tabličnih kalkulatora temeljnih na računalstvu u oblaku, tablični kalkulatori postali su računalne platforme sposobne za složene analize i modeliranje.

Istraživanja i radovi temeljeni na različitim aspektima razvoja i korištenja tabličnih kalkulatora dosljedno ukazuju na učestalost pogrešaka u tabličnim kalkulatorima, u rasponu od jednostavnih pogrešaka kod unosa podataka do složenih logičkih pogrešaka u formulama i izračunima. Iako ne postoji univerzalno prihvaćena taksonomija pogrešaka u tabličnim kalkulatorima, jedna od često korištenih klasifikacija pogrešaka u tabličnim kalkulatorima, razlikuje kvantitativne i kvalitativne pogreške kao temelj za daljnja istraživanja. Kvantitativne pogreške izravno utječu na netočne numeričke vrijednosti ili logičke pogreške u trenutnim rezultatima prikazanim u tabličnim kalkulatorima. Kvalitativne pogreške ne utječu na trenutni prikaz podataka i rezultate u tabličnim kalkulatorima, već predstavljaju skup objedinjenih pogrešaka koje su rezultat lošeg dizajna, netočnih pretpostavki korištenih u razvoju programskog modela ili strukturne nedostatke koji smanjuju kvalitetu tabličnih kalkulatora tijekom cijelog životnog ciklusa korištenja. Kvalitativne pogreške povećavaju vjerojatnost budućih kvantitativnih pogrešaka i negativno utječu na održavanje ili ispravljanje pogrešaka u tabličnim kalkulatorima. Posljedice takvih pogrešaka u tabličnim kalkulatorima mogu uzrokovati nepouzdane financijske izvještaje, netočne znanstvene analize i krivo donošenje poslovnih odluka koje u konačnici mogu uzrokovati potpuni kolaps i bankrot organizacija.

Recentna istraživanja u području tabličnih kalkulatora usredotočena su na metode pronalaženja i izbjegavanja pogrešaka. Metode detekcije pogrešaka temelje se na manualnim i automatskim metodama za otkrivanje pogrešaka. Manualne metode objedinjuju revizije tabličnih kalkulatora i testiranja koje provode eksperti iz različitih poslovnih područja.

Automatske metode detekcije pogrešaka temelje se na alatima za analizu koda i tehnikama za vizualizaciju i otkrivanje anomalija u podacima i izračunima. Preventivne metode za izbjegavanje pogrešaka temelje se na primjeni najboljih praksi u dizajnu i razvoju tabličnih kalkulatora, uključujući modularnost u dizajnu kompleksnih izračuna, te korištenje novih funkcionalnosti i programskih jezika u modernim tabličnim kalkulatorima.

Istraživanje prezentirano u ovom radu temelji se na metodi automatiziranog osiguranja kvalitete tabličnih kalkulatora u višekorisničkim okruženjima. Konkretno, ovaj rad prikazuje koncept i strukturu novog protokola ABAC4S (engl. Attribute Based Access Control for Spreadsheets) za automatizirano osiguranje kvalitete tabličnih kalkulatora u višekorisničkim okruženjima. Novi ABAC4S protokol za automatizirano osiguranje kvalitete programa tabličnih kalkulatora objedinjuje obje metode u fokusu istraživanja kvalitete tabličnih kalkulatora: metode pronalaženja (detekcije) i metode izbjegavanja (prevencije) pogrešaka u tabličnim kalkulatorima. U sklopu istraživanja prezentiranog u ovom radu, konceptualni model ABAC4S protokola formalno je verificiran metodom provjere modela (engl. Model Checking).

Ključni pojmovi: tablični kalkulatori, pogreške u tabličnim kalkulatorima, kontrola pristupa temeljena na atributima, metoda provjere modela

Contents

List of Figures

No. Figure name Page

List of Tables

No. Table Name Page

List of Acronyms

A*GED	A-Star Graph Edit Distance
ABAC	Attribute-Based Access Control
ABAC4S	Attribute-Based Access Control for Spreadsheets
ACL	Access Control List
COM	Component Object Model
CTL	Computation Tree Logic
DAC	Discretionary Access Control
DF-GED	Depth-First Graph Edit Distance
DLL	Dynamic Link Library
DSR	Design Science Research
ECMA	European Computer Manufacturers Association
ERP	Enterprise Resource Planning
EuSpRIG	European Spreadsheet Risks Interest Group
FLAME	Formula Language Model for Excel
GED	Graph Edit Distance
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
LLM	Large Language Model
MAC	Mandatory Access Control
NIST	National Institute of Standards and Technology
NTC	Negative Temperature Coefficient
QA	Quality Assurance
RBAC	Role-Based Access Control
SDLC	Software Development Life Cycle
SpACE	Spreadsheet Auditing for Customs and Excise
UML	Unified Modeling Language
VBA	Visual Basic for Applications
XLS	Microsoft Excel Format Spreadsheet File
XLSX	Microsoft Excel Open XML Format Spreadsheet File
XML	Extensible Markup Language

Spreadsheet Terms and Concepts

To ensure consistency of the research presented in this thesis, key spreadsheet terms and concepts are defined. More detailed and comprehensive insights into other spreadsheet terms and concepts are provided in later chapters of this thesis.

Spreadsheet Resource

Key concepts presented in this research relate to representation of spreadsheets as a collection of spreadsheet resources. Intentionally, the term Resource is used in contrast to the terms Object or Entity to avoid misinterpretation with Object Oriented Programming or Relational Database Systems. Modern spreadsheets programming resembles many concepts typically associated with Object Oriented Programming; however, spreadsheet resource should not be perceived as object in Object Oriented Programming. Spreadsheet resources are a more generic term representing multiple spreadsheets building components that are available to users either as built in functionality within modern spreadsheets or custom spreadsheet resources that users can construct with multiple different programming paradigms and tools.

Spreadsheet Program

Spreadsheet program consists of all spreadsheet resources utilized by spreadsheet users to solve user's problem. Most commonly, spreadsheet programs are constructed with built-in spreadsheet calculation directives and data that are needed to properly specify spreadsheet formulae. Some spreadsheet programs are simple one-time calculations, while others are used for complex tasks and computational modeling. Enterprise spreadsheet programs are developed to support organizational processes and support collaborative work of many spreadsheet users.

Spreadsheet Application

Spreadsheet application is a commercial application (like Microsoft Excel, Google Sheets, or LibreOffice Calc) supporting development and execution of spreadsheet programs. Modern cloud-based spreadsheet applications can be considered as powerful spreadsheet runtime environments empowering users to develop spreadsheet programs with multiple programming languages. One of the most popular spreadsheet applications is Microsoft Excel that can be used either as a standalone application or powerful cloud-based spreadsheet application.

Spreadsheet

In context of this research thesis, the term spreadsheet is used to represent union of spreadsheet applications and spreadsheet program. Spreadsheet is a commercial spreadsheet application executing user's spreadsheet programs. Even though users can build their own graphical user interfaces to interact with their spreadsheet programs, in most cases powerful built in spreadsheet application interface is utilized for interaction with spreadsheet programs.

Spreadsheet Formula Language

A spreadsheet formula language is a specialized programming language used within spreadsheet applications to construct spreadsheet programs, perform calculations and manipulate data. While often considered simpler than general-purpose programming languages, modern spreadsheet formula languages are very powerful, and they can theoretically simulate any computer algorithm. The recent addition of LAMBDA function in Microsoft Excel allows users to define reusable functions with recursions and function composition, making Excel a functional programming environment. Regardless of popularity, commercial vendors have never published official and concise grammar to facilitate parsing and analysis of spreadsheet formula language. One of the most complete grammar specifications of spreadsheet formula language has been published by Aivaloglou et al. [51].

Spreadsheet Error

According to the IEEE Standard Classification for Software Anomalies [52] an "error" is a misapprehension on side of the one developing a software caused by a mistake or misconception occurring in the human thought process. A "fault" is the manifestation of an "error" within a software which may be causing a "failure". A "failure" is the deviation of the observed behavior of the software from the expectations. However, researchers and spreadsheet practitioners are not consistent with IEEE Standard Classification for Software Anomalies [52] and "fault" and "error" are often used as synonyms. This research thesis will continue with tradition of many spreadsheet researchers and if not specifically written, term "error" will be used as synonym for "error" or "fault".

1. Research Methodology

The research methodology in this thesis follows the Design Science Research (DSR) approach [35]. Activities and phases conducted during defined research methodology are visually represented in Figure 1.

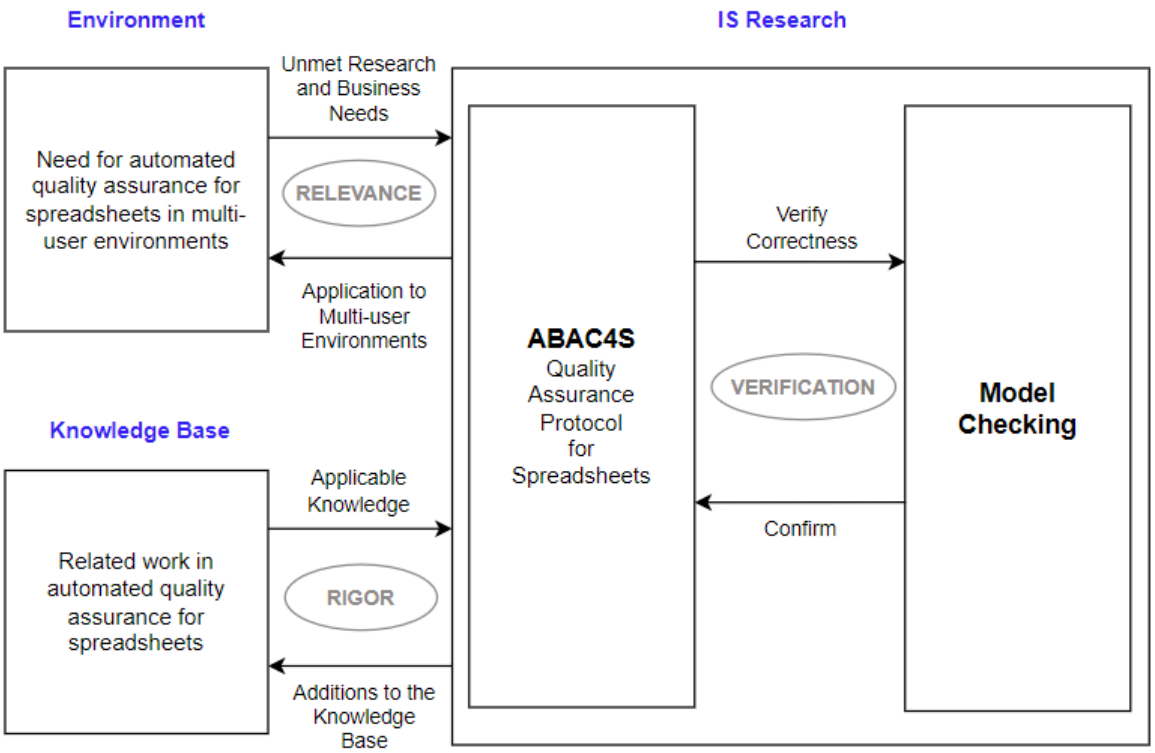

Figure 1. Reseach Methodology based on Design Science Research

DSR methodology is structured around three interconnected components or research phases that should jointly deliver design science research artefacts. Specifically, the application of DSR methodology to the research presented in this thesis resulted in the following method and research phases:

- **Environment:** The environmental part of the Design Science Research methodology has been defined in the Introduction and Motivation chapter of this thesis. In addition to the presented spreadsheet horror stories, automated quality assurance methods for finding (detection) and avoiding (prevention) spreadsheet errors have been presented. Two use cases with application of developed ABAC4S protocol for automated quality assurance of spreadsheets in multi-user environments have been demonstrated in this research thesis.

- **Knowledge Base:** This research journey has started with comprehensive literature review to understand existing knowledge base, as well as challenges and opportunities required to deliver effective Design Science Research artifacts. Specifically, literature review and discussions in this thesis have been focused on taxonomy of spreadsheet errors, automated detection of spreadsheet errors and controlled access for spreadsheet users in modern enterprises. Based on the existing knowledge base and literature review conducted, automated quality assurance for spreadsheets has been identified as active research domain with unmet research needs for finding (detection) and avoiding (prevention) spreadsheet errors in multi-user environments. Furthermore, design considerations for design science research artifact derived from existing knowledge base have been combined with requirements to address unmet research and business needs for automated quality assurance of spreadsheet programs in multi-user environments.
- **IS Research:** Information System Research part of methodology represents a core set of research activities presented in this thesis. Main deliverable of this research phase is novel ABAC4S protocol for automated quality assurance of spreadsheets in multi-user environments. The formal specification of novel ABAC4S protocol for spreadsheet will be provided with multi-faceted approach and combination of visual modeling, specification of protocol building blocks with algebraic data structures and direct graph representation. As part of the ABAC4S protocol specification, algorithm to detect and resolve conflicts between ABAC4S access rules is presented and formally evaluated for correctness with model checking approach.

Aligned with the Design Science Research approach and requirements to develop novel ABAC4S automated quality assurance protocol for spreadsheets in multi-user environments, the first research goal for research presented in this thesis has been formulated:

- RG1: Develop formal description of ABAC4S protocol for automated quality assurance of spreadsheets, capable of controlling user's interaction with spreadsheets in multi-user environments.

Research focused on formal description of ABAC4S protocol for automated quality assurance has been inspired by conceptual model for measuring complexity of spreadsheets [48] and spreadsheet representation as collection of resources [6]. The first research goal is the

basis for the second research goal, in that it provides specifications of the ABAC4S protocol for automated quality assurance of spreadsheets that can be formally verified. The next step in research presented in this thesis is evaluation of ABAC4S protocol correctness with a model checking approach, which leads to second research goal:

- RG2: Evaluate correctness property of the proposed ABAC4S protocol for automated quality assurance of spreadsheets with model checking approach.

To address the second research goal, ABAC4S protocol for automated quality assurance of spreadsheets will be modeled as a system with a set of states and transition relations that specify the behavior of the system. Aligned with the above research goals, the following research hypothesis is formulated:

- H0: ABAC4S protocol for automated quality assurance of spreadsheets in multi-user environments, correctly evaluates spreadsheet state changes for given user's roles.

NuSMV model checker has been utilized to verify correctness property of the proposed ABAC4S protocol for automated quality assurance of spreadsheets in multi-user environments [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. Model checking tool explores all possible traces in search of counterexample where desired property formulated with research hypothesis is not satisfied.

Research presented in this thesis followed an iterative approach through experimentation, simulation and model checking to refine outcomes of the research phases and fine tune characteristics of developed ABAC4S protocol. Initial results of the ABAC4S protocol verification with model checker and provided counterexamples were instrumental for ABAC4S protocol redesign, including refinements of rules and algorithm for conflict resolution within set of user's access rules.

2. Introduction and Motivation

Spreadsheets are structured around a powerful and intuitive graphical user interface represented as collections of two-dimensional grids. Each two-dimensional grid is a collection of cells, uniquely identified by address represented with row and column index within the respective grid. Spreadsheet computation capabilities are constructed with formulae placed in cells depending on user's need and spreadsheet program functionality. Spreadsheet computations are event based, and changes introduced by an actor into the spreadsheet program are immediately evaluated and computed. Spreadsheet actors are users working interactively with spreadsheets or external systems capable of accessing spreadsheet programs through interfaces. Commercial spreadsheets have a broad range of embedded formulae and support various mechanisms for building extensions such as user defined formulae, customizations of the spreadsheet interface and embedding new computational modules constructed with external procedural, object oriented or functional programming languages.

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]. 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]. The following are summaries of several spreadsheet horror stories that caused reputational and financial impacts on individuals, organization and institutions.

16000 UK Covid-19 test results lost for weeks

The issue in this case is that PHE's (Public Health England) own developers picked an old file format to develop spreadsheet program – known as XLS. Consequently, each template could handle only about 65000 rows rather than the one million-plus rows that Excel is actually capable of. When the limit of an old file format was reached, further cases were simply left off.

To handle the problem, PHE is now breaking down the test result data into smaller batches to create a larger number of Excel templates.

Scientists rename human genes to stop Microsoft Excel from misreading

The bioinformatics community decided it was easier to change gene symbols than changing peoples' habits. When scientists want to process numerical data, spreadsheets are often used to import/export numerical data and perform further analysis. Uncontrolled process for numerical data formatting and data type designation, caused wrong data type inference and unwanted data transformations.

Unofficial spreadsheets land Italian pharma plant with regulatory warning

Regulatory warning was issued to Italian pharma company due to the use of “unofficial” and uncontrolled spreadsheet on a shared network drive.

The Norwegian Sovereign Wealth Fund's $92 Million Excel Error

In 2024, Norway's sovereign wealth fund revealed that it had lost roughly $92 million, on an error relating to how it calculated its mandated benchmark. Calculation error was discovered in the composition of the index, because an incorrect date was manually entered in the spreadsheet.

Excel formula error inflated myocarditis statistics

According to UK National Health Service, the initial claim of 8% of people affected by heart issues have been corrected to 0,01%, due to excel formula error that occurred during the process of simplifying the data into a pivot table. The value that was displayed was the sum total of a numeric value within the raw data, specifically a row count, as the years progressed the row count increased meaning the sum was greater. The value displayed should have been a count and not a sum.

Spreadsheet error led to Edinburgh hospital opening delay

Human errors in spreadsheet with the specification for air flow in critical care rooms have been discovered prior to hospital opening day. Independent checks found that critical care rooms were operating with the wrong air flow. Remedial work to correct identified spreadsheet error with the wrong air flow has been estimated to 16 million GBP.

Private data leakage in spreadsheet “hidden” column

Global aerospace firm Boeing reported in 2017 that a company employee mistakenly emailed a spreadsheet full of employee personal data to his spouse. The spreadsheet, sent to provide the employee’s spouse with a formatting template, contained the personal information of roughly 36,000 other Boeing employees, including Social Security numbers and dates of birth, in hidden columns.

Spreadsheet error costs Tibco shareholders $100M

Tibco Software shareholders will be getting $100 million less than originally anticipated from the company’s more than $4 billion sale to Vista Equity Partners because of a spreadsheet error that overstated Tibco’s equity value. According to a regulatory filing, Goldman Sachs, which is advising Tibco on the deal, used the spreadsheet in calculating that Tibco’s implied equity value was about $4.2 billion.

$1,1 Billion “Honest Mistake” error in financial result statement

Fannie Mae, the mortgage company, made a $1.1 billion mistake in a spreadsheet during adoption of new accounting standards. Fannie Mae senior vice president for investor relations, said: “There were honest mistakes made in a spreadsheet used in the implementation of a new accounting standard.”

Ticket mishap at the London 2012 Olympics

Booking tickets for the 2012 Olympic games in London was done in rounds. Thousands of tickets were unclaimed when second-round ticket sales of the synchronized swimming competition began. After realizing something was off, it was revealed that an employee for the ticketing company made the simple mistake of typing “2” instead of “1”, resulting in 20000 seats going on sale instead of 10000. Fortunately, this error was identified early on and those who purchased tickets to what essentially amounted to non-existent seats were able to exchange them for other events within the games. Some customers even saw this as a blessing in disguise; they were upgraded from their original purchase and ended up attending higher profile competitions which would not have been possible otherwise.

MI5 makes 1,061 bugging errors

UK secret intelligence MI5 wrongly bugged more than 1000 phones. MI5 stated that a spreadsheet formatting error caused formatting to be applied on telephone numbers ending in 000 rather than the actual last three digits.

Research consistently demonstrates that spreadsheet errors are omnipresent, with studies revealing error rates in a substantial percentage of models, ranging from simple data entry mistakes to complex logical flaws in formulae. Based on research into spreadsheet errors, several taxonomies have been developed over time to classify different types of errors. While there isn't one universally adopted taxonomy that supersedes all others, researchers like Panko [10] and Rajalingham [12] classified spreadsheet errors into qualitative and quantitative errors. Quantitative errors directly lead to incorrect numerical values or logical flaws in the spreadsheet program direct results. Qualitative errors do not immediately produce incorrect numerical values but represent poor design practices, incorrect assumptions, or structural flaws that degrade the spreadsheet's quality during the lifetime of spreadsheet. Qualitative errors increase the likelihood of future quantitative errors, or make the spreadsheet difficult to understand, maintain, or debug. The impacts of such errors can be severe, leading to flawed financial forecasts, incorrect scientific analyses, misguided business decisions, legal liabilities, and even corporate collapses [4].

To address challenges associated with primarily qualitative spreadsheet errors, research in spreadsheet quality has focused on methods for finding (detection) and avoiding (prevention) spreadsheet errors. Detection methods range from manual auditing and peer review to more sophisticated techniques involving code analysis tools, testing-based techniques and data visualization for anomaly detection. Prevention strategies emphasize best practices in spreadsheet design including modularity, consistency with formatting and naming conventions and refactoring of spreadsheet formulae with modern structured references and named objects.

Furthermore, the development and application of formal quality frameworks and system development methodologies for spreadsheets follow proven and established development principles from general software engineering practices. These frameworks promote structured approaches during the whole life cycle of spreadsheet programs, with robust version control, user validation, static and dynamic testing, and the implementation of internal controls.

Based on existing knowledge base and literature review conducted, automated quality assurance for spreadsheets is active research domain with unmet research needs for qualitative spreadsheet errors in multi-user environments. Research presented in this thesis will define novel ABAC4S protocol for automated quality assurance of spreadsheet programs. Novel

ABAC4S protocol for automated quality assurance of spreadsheet programs uniquely addresses both methods in focus of quality assurance research, finding (detection) and avoiding (prevention) of spreadsheet errors and quality issues. The formal specification of novel ABAC4S protocol for spreadsheet will be provided with multi-faceted approach and combination of visual modeling, specification of protocol building blocks with algebraic data structures and direct graph representation. A multi-faceted approach allows clear communication of research deliverables to thesis readers and other researchers focused on exciting research related to spreadsheets. As part of the ABAC4S protocol specification, novel algorithm to detect and resolve conflicts between ABAC4S access rules is presented and formally evaluated for correctness with model checking approach. Detailed steps and research journey from original ABAC4S protocol idea to formal specification of the protocol in applicable model checking language is presented in this thesis. Finally, as part of the research presented in this thesis, the use cases of ABAC4S protocol evaluation within IT department and analytical laboratory are presented with documented ABAC4S access rules derived from organizational processes and users job descriptions.

3. Related Work

The tremendous success of spreadsheets and impact of spreadsheet errors triggered significant interest of the research community. In this thesis, I followed approach to literature review presented by Powell et al. [11]. Rather than give a chronological account of the literature on the spreadsheet research, discussion in this thesis is focused on taxonomy of spreadsheet errors, automated detection of spreadsheet errors, and controlled access for spreadsheet users in modern enterprises.

3.1. 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 of the above presented classification, Powell at al. [11] noted that this proposed classification does not consider context of spreadsheet use and how each error was committed.

The taxonomy of errors for spreadsheets developed by Rajalingham et al. [12] is one of the first attempts that introduced different spreadsheet user roles. Hierarchical view of taxonomy of errors for spreadsheets developed by Rajalingham et al. [12] is presented in Table 1.

Table 1. Taxonomy of spreadsheet errors

A. SYSTEM-GENERATED
B. USER-GENERATED
    a. QUANTITATIVE
        i. ACCIDENTAL
            1. DEVELOPER (workings)
                a. Omission
                b. Alteration
                c. Duplication
            2. END –USER
                a. DATA INPUTTER (Input)
                    i. Omission
                    ii. Alteration
                    iii. Duplication
                b. INTERPRETER (output)
                    i. Omission
                    ii. Alteration
                    iii. Duplication
        ii. REASONING
            1. DOMAIN KNOWLEDGE
                a. REAL-WORLD KNOWLEDGE
                b. MATHEMATICAL REPRESENTATION
            2. IMPLEMENTATION
                a. SYNTAX

b. LOGIC

C. QUALITATIVE
   a. SEMANTIC
      i. STRUCTURAL
      ii. TEMPORAL
   b. MAINTAINABILITY

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. Quantitative errors directly lead to incorrect numerical values or logical flaws in the spreadsheet program direct results. Sub-categories of quantitative spreadsheet errors often include the following:

- Mechanical Errors: Simple slips or mistakes, such as typos, incorrect data entry, or incorrect cell references due to pointing errors.
- Logic Errors: Flaws in the underlying reasoning or formula construction, where the formula itself is incorrect for the intended calculation. Examples of logical errors are using the wrong spreadsheet function, incorrect parameters during function invocation, incorrect operator, or flawed algorithm.
- Omission Errors: Omission errors occur when required data or calculation formulas are accidentally left out, leading to incomplete or inaccurate results. This can also include misinterpretation of the problem to be modeled.
- Accidental Errors: Accidental errors occur as result of user's negligence, stress or oversight, often leading to immediate issues with spreadsheet programs.
- Reasoning Errors: Reasoning errors stem from wrong or flawed users understanding of the problem or the spreadsheet's functionality, leading to incorrect intentions or actions.

Qualitative errors do not immediately produce incorrect numerical values but represent poor design practices, incorrect assumptions, or structural flaws that degrade the spreadsheet's quality during the lifetime of spreadsheet. Qualitative errors increase the likelihood of future quantitative errors, or make the spreadsheet difficult to understand, maintain, or debug. Sub-categories of qualitative spreadsheet errors often include the following:

- Structural Errors: Structural errors represent non-compliance or gaps to spreadsheet development standards or best practices. Typical structural errors found in spreadsheets are flaws in the design or layout of the model, incorrect or ambiguous labeling, poor formatting, or issues that make the spreadsheet hard to navigate or understand. Hard-coding numbers directly into formulae instead of referencing named reference is a common example.
- Temporal Errors: Temporal errors relate to using outdated or non-current data. One additional challenge when spreadsheet programs output questionable data is inappropriate use of volatile functions in Microsoft Excel spreadsheet. Volatile functions update dynamically whenever Excel recalculates, even if no changes are introduced to the spreadsheet program. This includes functions like `TODAY()`, `NOW()`, `RAND()`, `RANDBETWEEN()`, and certain versions of `CELL()`, `OFFSET()`, `INDIRECT()`, and `SUMIF()` depending on their arguments. The resulting value of the volatile functions cannot be assumed to be the same from one moment to the next even if none of its arguments (some functions do not require input arguments) have changed. In addition to frequent cause of temporal errors when used inappropriately, volatile functions might cause degradation in spreadsheet performance due to frequent recalculations. For example, the use of function `TODAY()` in a cell will always display the current date in a cell and will update to tomorrow's date if the spreadsheet is opened tomorrow. In contrast, the use of function `DATE(2025, 06, 09)` in a cell will always show *June 09, 2025* (depending on cell formatting), regardless of when the spreadsheet is opened or recalculation is triggered by user. It is important to note that system time information used for date functions in spreadsheets is primarily derived from device's clock and time settings in case of standalone spreadsheet applications and from browser settings in case of cloud-based spreadsheets.
- Maintainability Errors: Maintainability errors stem from overcomplex spreadsheet programs and design issues that make the spreadsheet difficult to update, audit, or maintain in the future.
- Semantic Errors: Semantic errors occur when a spreadsheet program's code is correct from a syntax standpoint but produces unpredictable or incorrect results due to flaws in the underlying algorithm or logic.

While there isn't one universally adopted taxonomy of spreadsheet errors that supersedes all others, core distinction between quantitative and qualitative errors remains central idea for research. Recent upgrades to spreadsheet error taxonomies tend to offer more granular description and sub-classifications for quantitative and qualitative errors.

3.2. 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. As of today, Microsoft Excel will notify users with the following errors as result of invalid formula evaluation:

- #DIV/0
- #N/A
- #NAME?
- #NULL!
- #NUM!
- #REF!

- #VALUE!
- #GETTING_DATA
- Anything else (returns #N/A)

Due to the constant innovation in spreadsheets, the above list will have to be reviewed and updated to match new functionalities in recent versions of commercial spreadsheets. For example, introduction of lambda function in the latest development version of Microsoft Excel spreadsheet allows users to generate custom reusable functions and use recursions without the need of external programming languages. Recursions in lambda formula will require review and potential redesign of existing formula evaluation errors.

Aurigemma and Panko conducted a study to evaluate two commercial spreadsheet static analysis tools, both with each other and human auditors [37]. Overall results of automated static analysis tools did not outperform human auditors. The 33 human inspectors found 54 out of 97 errors, while only 5 errors were tagged by two commercial tools. These results cannot be generalized to wider context of spreadsheet use and all stages of spreadsheet lifecycles. However, results presented, and overall performance of automated static analysis tools indicated direction for future research.

In the second study, the auditing procedure used by HM Customs and Excise was described by Butler [38]. This procedure is hybrid and includes manual activities performed by expert auditors, as well as automated activities performed with commercial software tool (SpACE). The procedures work as follows. First, the auditor identifies the chain of cells from inputs to end result and uses the software tools (SpACE) to follow the chain of dependent cells so that the key formulas can be checked. Then the auditor checks the original formulas that were used to copy related formulas, and checks that the copies are correct. Main contribution of automated software tool used in this procedure is to speed up the overall review process and assist human auditor with high-risk cells.

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. The 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].

3.3. Access Control for Spreadsheets

Access control and authorization are key components of information technology systems in multi-user environments [23]. Focus of researchers have been around modelling different access control systems, evaluation and comparison of access control models deployed to various technical and operational environments, and formal verification of access control models in context of specific algorithms and protocols. Following is the summary of common access control models that have been analyzed for their suitability in development of access control protocol for spreadsheets [39].

3.3.1. Discretionary Access Control (DAC)

Discretionary Access Control (DAC) is access control model built with three major components – objects, subjects, and permissions. DAC allows owners (subjects) to control permissions to their objects and is commonly implemented with Access Control List (ACL). DAC is implemented as an integral part of many information technology systems, such as operating systems and databases. DAC has been part of commercial spreadsheet implementation for decades and allows spreadsheet owners to control user's access to specific cells or worksheets. Downs et al. [40] studied issues with DAC, and this type of access control does not allow granularity for different roles, centralized administration with access policy and

monitoring data flow becomes almost impossible as the spreadsheet program grows in complexity.

3.3.2. Mandatory Access Control (MAC)

Mandatory Access Control (MAC) is access control model managed in centralized manner and is built with four key components – a set of objects, a set of subjects, permissions, and security level. Even though MAC allows centralized policy management, it is very complex to implement this type of access control on all spreadsheet resources due to mandatory security level assigned to both subjects and objects. In addition, security levels specified in traditional MAC have been considered by researchers as antiqued [41]. Due to mandatory security level for subjects and objects, MAC is very complex for implementation and does not fit to technical and organizational dynamics of modern enterprises.

3.3.3. Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) is access control model based on following key components – subjects, roles, permissions, actions, operations, and objects. In context of RBAC, role means a group of permissions to use object(s) and perform certain action(s). Only designated administrators have the right to control system security and manage roles assigned to users. RBAC implementation has been studied in hospital management where roles allow modeling complex relationships between doctors, nurses, and other hospital stakeholders [42]. However, modeling and maintaining roles for different spreadsheet user roles and groups is very complex, especially in dynamic organizations where business processes and corresponding user roles are rapidly changing.

3.3.4. Attribute-Based Access Control (ABAC)

Attribute-Based Access Control (ABAC) is access control model based on following three types of dynamic attributes – subjects, objects, and environments. User requests are resolved and determined based on subject attributes, objects attributes, environmental attributes as well as set of conditions specified by access policy. ABAC model is dynamic as it uses state of attributes at the time of access mode resolution.

Even though limited literature on the application of ABAC access control model to spreadsheets has been found, following ABAC access control model properties have been

instrumental for development of ABAC4S protocol for automated quality assurance of spreadsheets in multi-user environments:

- ABAC model is based on dynamic attributes, where object attributes fit to our proposed model of spreadsheet resources and corresponding attributes.
- Hierarchy of spreadsheet resources and objects can be modelled with set of 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 allows early prototype implementation as detective access control system. This minimizes impact on users and their interaction with spreadsheet interface.
- Complexity of ABAC model for spreadsheets depends on number of spreadsheet resource attributes.
- Dynamic nature of modern cloud-powered spreadsheets and extensions to spreadsheet formula language fits nicely to ABAC dynamic attribute concept. Potential new functionalities and modules added in cloud-powered spreadsheet can be integrated within existing ABAC concepts.

Unified metamodel of enterprise authorizations is summarizing existing models of access controls [23]. In addition, authors provided mapping between presented unified metamodel and ArchiMate tool that is frequently used in modern enterprises as architecture modeling language. List of generic metamodels for expressing different configurations of access models has been valuable starting point for this research and design of ABAC metamodel for spreadsheets.

ABAC modelling and implementation has been recognized by U.S. government as important access control modelling concept for large enterprises and federal information technology systems. National Institute of Standards and Technology (NIST) published in 2014 Special Publication 800-162 “Guide to Attribute Based Access Control (ABAC) Definition and Considerations” [24]. This publication provides definitions and considerations for using ABAC to improve information sharing and design of systems, while maintaining control of that information. Concepts and terminology for ABAC presented in this document have been instrumental for design of automated quality assurance for spreadsheets presented in this thesis.

4. ABAC4S Protocol

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 six 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 *Data Model* used for protocol implementation
4. The *Sequence Diagrams* with data flows during protocol execution
5. The *Encoding* (format) of ABAC4S protocol access rules
6. The *Processing Logic* of ABAC4S protocol, including algorithm for resolution of conflicts within access rules and determination of spreadsheet changes between different spreadsheet states. These rules guard the consistency of data flows and correctness of Service provided by the ABAC4S 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 algebraic and directed graph data structures and data flows suitable for translation to model checking tools and verification of correctness for protocol rules. ABAC4S protocol for automated quality assurance of spreadsheets should control evolution of spreadsheet programs according to user's access rules and structured criteria defined for spreadsheet resources. As such, ABAC4S protocol is managing behavioral and structural quality criteria of spreadsheet programs defined with access rules.

4.2. 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 for automated quality assurance of spreadsheets 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 ABAC4S protocol processing logic, while implementation for protocol execution on commercial spreadsheets is left for specific deployment scenarios.

4.3. ABAC4S Protocol Data Model

A data model refers to an abstract representation of data structures that are used to organize and manage data in the proposed ABAC4S protocol for automated quality assurance of spreadsheets. First, ABAC4S protocol building blocks have been defined with visual conceptual models to facilitate communication and understanding of data requirements for ABAC4s protocol. Class diagrams from Unified Modeling Language (UML) have been used to define a modern cloud-based spreadsheet conceptual model as extension to spreadsheet conceptual model introduced by Retschenhofer et al. [48]. UML class diagrams will be used to define conceptual models of ABAC4S access rules with links to the associated set of spreadsheet resource attributes. Based on provided visual conceptual models for spreadsheets, further refinement of proposed conceptual model will be provided with directed graph representation of spreadsheet resources. This combination of UML and directed graph representation of spreadsheets is used in later stages of this research to perform formal verification of proposed ABAC4S protocol and assess protocol correctness with model checking approach.

4.3.1. Spreadsheet Conceptual Model

Conceptual models of spreadsheet resources and associated attributes are visually presented in Figure 2. and Figure 4.

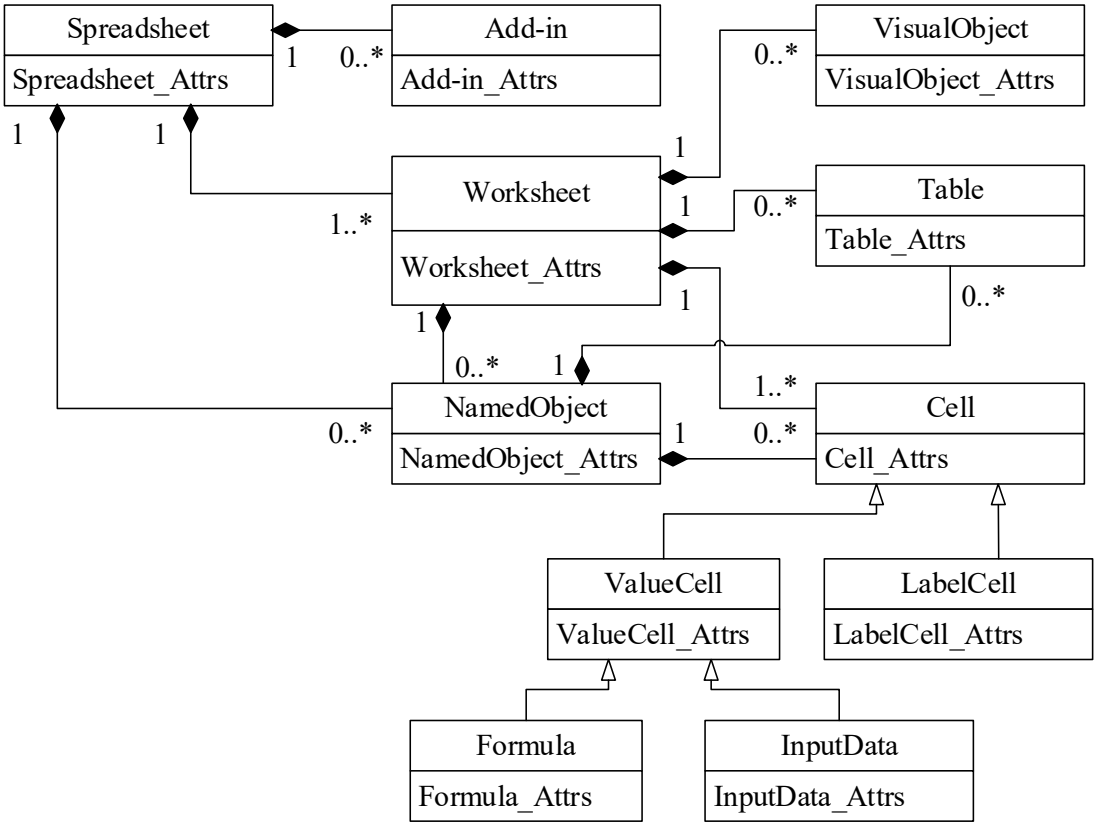

Figure 2. Conceptual model of spreadsheet resources and associated attributes.

Design considerations in proposed ABAC4S protocol are structured around resources that constitute modern cloud-powered spreadsheet. Spreadsheet resources are building blocks for spreadsheet programs and are manipulated by spreadsheet users or change their state during lifecycle of spreadsheet as result of spreadsheet program execution. Spreadsheet resources and their attributes are bounded with ABAC4S rules and permissible actions performed by spreadsheet users. The following provides a description of key spreadsheet resources and corresponding classes in spreadsheet conceptual model diagrams:

- The class *Spreadsheet* represents spreadsheet program. This is the root class and container in the presented metamodel for other spreadsheet resources. Spreadsheet programs that are instantiated from this class are stored in proprietary file formats on personal computers. The most popular format used by Microsoft Excel is Microsoft Excel Open XML Format (XLSX). It's an Extensible Markup Language (XML) file format based on ECMA-376 Office Open XML standard [45]. XLSX file format uses standard ZIP compression to store data in a more efficient and accessible way, readable on different operating systems and

computer architectures. XLSX offers better data integrity, improved compatibility, and a more structured approach to data storage compared to the older XLS file format.

- The class *Worksheet* represents a blueprint for instantiating worksheets. Worksheets are a primary working area and key resource within each spreadsheet represented as a two-dimensional collection of cells organized into rows and columns. In modern cloud-based spreadsheets, worksheets are containers for other spreadsheet resources, such as visual objects, tables and other custom-built resources such as user defined functions or computational modules constructed with external programming languages. Depending on spreadsheet user preferences, worksheets can be used as scoping boundaries for custom user defined functions and named objects.
- The class *NamedObject* represents various resources in modern spreadsheets that are controlled by users through Named Manager functionality. This naming convention is used in Microsoft Excel 365 product, but other cloud-based spreadsheets offer similar functionality. In modern cloud-based spreadsheets, class *NamedObject* represent all spreadsheet resources constructed with either composition of internal spreadsheet functions, or with external programming languages. With the recent introduction of `LAMBDA` functions and collection of supporting functions (`MAP`, `REDUCE`, `SCAN`, `MAKEARRAY`, `BYROW`, `BYCOL`, `ISOMITTED`), Microsoft Excel is additionally empowered for computational tasks previously reserved for plugins or scripting with embedded macro language [43]. Great demonstration of new Microsoft Excel capabilities and powerful development strategies with custom user defined functions based on Excel formulas has been provided by Bartholomew at EuSpRIG conference [44]. Customized user' spreadsheet resource created as instance of *NamedObject* class can be scoped to broader spreadsheet workbook or limited to the worksheet. Reusable software components built with external programming languages are tightly integrated into modern spreadsheets, empowering spreadsheet users to develop models quicker with fewer errors [46]. For example, the new `=PY()` function introduced in Microsoft Excel spreadsheet acts as a bridge, letting users write Python code directly in Excel cells [47]. Python in Excel is compatible with existing tools and libraries for charting and numerical analysis. Among many exciting new features that Python in Excel offers to users is the ability to create and dynamically control complex tabular and visual objects directly from python code. This functionality was previously available only through Excel predefined toolbar.

- The class *Add-in* represents an external program that extends the functionality of the spreadsheet with new features and commands. Resources created as instance of *Add-in* class are the oldest mechanism available in spreadsheets for building custom extensions. Microsoft popularized Add-ins in early versions of Excel with proprietary architectures based on Dynamic Link Library (DLL) and Component Object Model (COM). COM is specification for creating reusable software components that can interact with each other. Traditional Add-ins built on COM specifications are bound to client versions of Excel spreadsheets and must be installed separately. In modern spreadsheets, lambda functions are essential for the creation of reusable software components, empowering spreadsheet developers to develop models quicker with fewer errors [46]. Another new feature added to Excel is native support for Python programming language [47]. Powerful Python computational engine is embedded in Excel and users can integrate Python language code with existing formula language at the level of cell. Python in Excel is compatible with existing tools and libraries for charting and numerical analysis. Among many exciting new features that Python in Excel offers to users is the ability to create and dynamically control complex tabular and visual objects directly from python code. This functionality was previously available only through Excel predefined toolbar.
- The class *Table* represents Excel named table resources that could be added to the worksheet through spreadsheet application interface. However, after manual creation of named table, this spreadsheet resource can be controlled directly from spreadsheet formula language with structured references. Microsoft introduced structured references as part of named tables functionality in Excel 2007. These references, which use column names instead of cell references in formula language, were a key feature of the new named table format. Structured references are powerful extensions to formula language and allow spreadsheet users to reference custom tables in their entire code, with improved readability and maintainability of spreadsheet programs.
- The class *VisualObjects* represents spreadsheet visual objects that could be added to the worksheet through spreadsheet application interface. These graphs and charts are visual representations of worksheet data that help to illustrate trends, pinpoint patterns, and make comparisons within data. All commercial spreadsheets include countless options for generating charts and graphs, including bar charts, line charts and other more complex data visualization techniques. Charts and graphs are powerful spreadsheet tools and help communicate clearly and efficiently, especially for large and complex data sets.

- The class *Cell* represents key spreadsheet resource and smallest unit of data storage in a spreadsheet. A cell is a single rectangular area where data is entered and stored, formed by intersection of a column and a row. In commercial spreadsheets there are two ways for referencing cells. In A1 Reference Style, cells are referenced using a combination of a column letter and a row number, such as A1, B1, or D10. Relative cell references change during copy operation to other cells. For example, if A1 cell reference is copied from original location to new location with relative distance of 1 column and 1 row to original cell location, the new cell reference will be B2. Spreadsheet formula language uses special syntax to control absolute and relative cell references. Dollar sign ($) is used to fix the row or column, preventing cell reference from changing when copied or filled. For example, $A1 fixes column A, and A$1 fixes row 1 during copy of filling operation. Absolute cell reference, immune to changes during copying and filling operations, is achieved with A1 cell reference. There's also an alternative, R1C1 Reference Style, where cells are referenced by row and column numbers preceded by "R" and "C" respectively, for example, R1C1, R2C1, or R10C4. Relative cell referencing for R1C1 Reference style is achieved with numbers in square brackets to indicate the offset from the current cell. For example, R[1]C[1] refers to the cell 1 rows below and 1 column to the right of the cell containing the formula. Negative numbers indicate moving up or left, and positive numbers indicate moving down or right. A1 Style Reference is preferable cell reference style for majority of spreadsheet users. However, R1C1 Style reference offers several advantages, particularly when working with relative cell references and formulas that are intended to be copied or filled. For example, a formula like R[-1]C[-1] (one row above and one column to the left) will remain the same when copied across multiple rows or columns. This contrasts with A1 Style Reference, where cell references might change based on the new cell's position. Additional benefits of R1C1 Style Reference can be useful when working with spreadsheet macros and Visual Basic for Applications (VBA) where cells are manipulated programmatically. Ranges are rectangular collection of cells defined by the cell reference of the top-left cell and the cell reference of the bottom-right cell separated by colon. For example, A1:C2 is a range containing 6 cells in total, with A1 as top-left cell and C2 as bottom-right cell. Spreadsheets allow 3D referencing to the same cell or range across multiple worksheets within a single spreadsheet workbook. For example, cells A1 on worksheets named “Sheet1”, “Sheet2” and “Sheet3” can use a 3D reference, without having to manually list each worksheet and cell in formula. Let’s say we have the following data in spreadsheet workbook:

Worksheet “Sheet1”: Cell A1 contains value “Banana”

Worksheet “Sheet2”: Cell A1 contains value “Apple”

Worksheet “Sheet3”: Cell A1 contains value “Orange”

To list values from each worksheet cell A1 in first column of worksheet “Sheet4”, we can use the following 3D reference as parameter to excel function `TOCOL()`:

```
=TOCOL(Sheet1:Sheet3!A1)
```

The `TOCOL()` function in Excel converts a multi-column array or 3D reference into a single column as visually presented in Figure 3. It takes an array, and optionally, arguments to ignore certain values or to scan by column. The function is useful for flattening data structures and organizing information into a more manageable format.

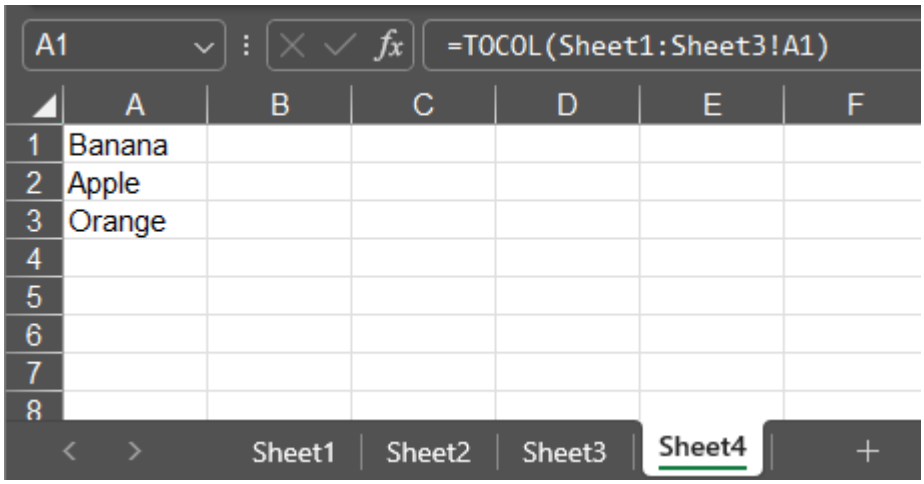

Figure 3. Example of 3D referencing in Excel Spreadsheet.

Spreadsheets allow cross-worksheet reference within formula or function in one worksheet that uses data or values from another worksheet within the same workbook. Cross-worksheet referencing creates dynamic links between cells or ranges across worksheets, so changes in one worksheet will automatically update in the other. For example, cross-worksheet reference `=Sheet1!A1` would refer to cell A1 on Sheet1 within the current spreadsheet workbook. As visually represented in metamodel in Figure 2., cells instantiated from class *Cell* might be of the type *ValueCell* or *LabelCell*. “Labeled cells” refers to cells that are used for identification of structured data, such as named tables or column headers. “Value cells” in presented metamodel might be of type *Formula* or *InputData*. “Input data

cells" generally refers to individual cells in a worksheet that contain the raw data being analyzed. These cells hold the raw information that spreadsheet programs used for computation and might be entered either manually by users or populated by automatic spreadsheet interfaces. The conceptual model for spreadsheet formula is presented in Figure 4.

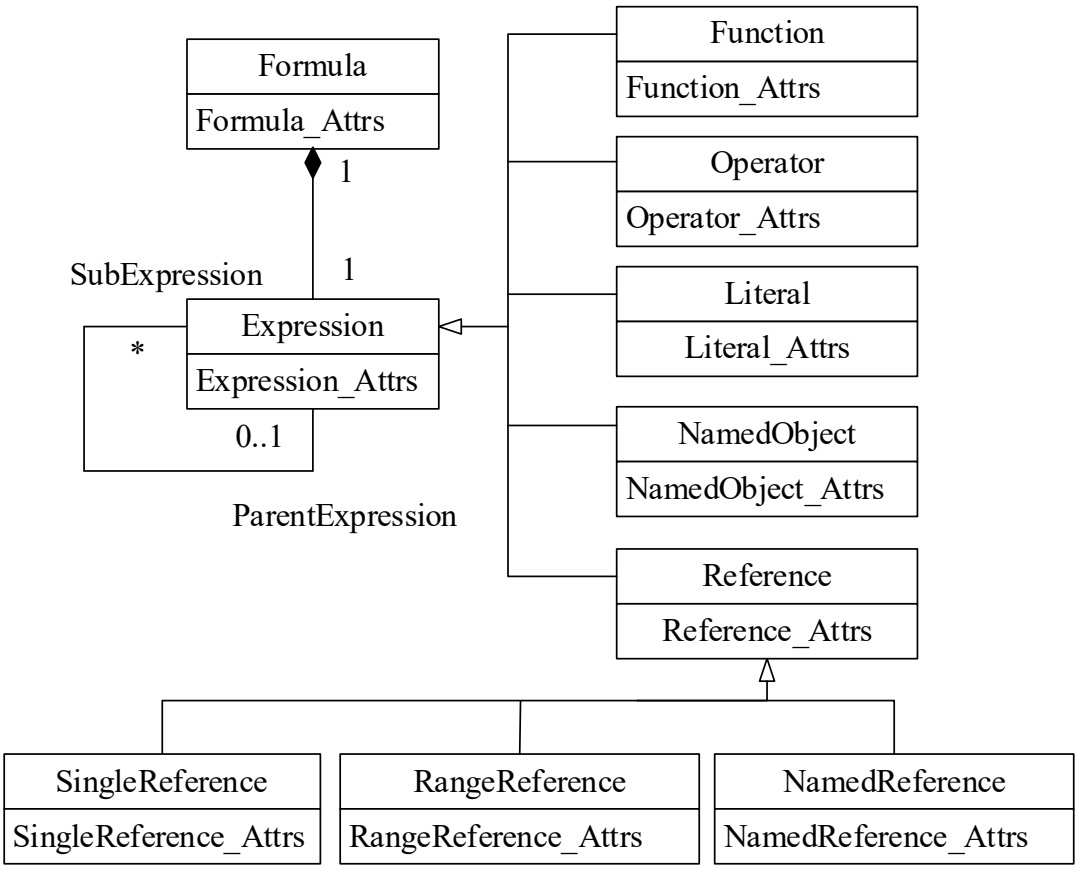

Figure 4. Spreadsheet formula conceptual model.

- The class *Expression* represents powerful spreadsheet formula language ability for nesting formulas. The original design introduced by Retschenhofer et al. has been extended to include *NamedObject* as parameters and *NamedReference* as one possible realization of cell references [48]. As represented in the provided conceptual model, expressions are constructed with functions, operators, literals, named objects and references. Expressions might also be composed of other expressions, thus allowing spreadsheet users to model complex computational problems.

4.3.2. ABAC4S Protocol Access Rules

The conceptual model of ABAC4S protocol access rules is visually represented in Figure 5.

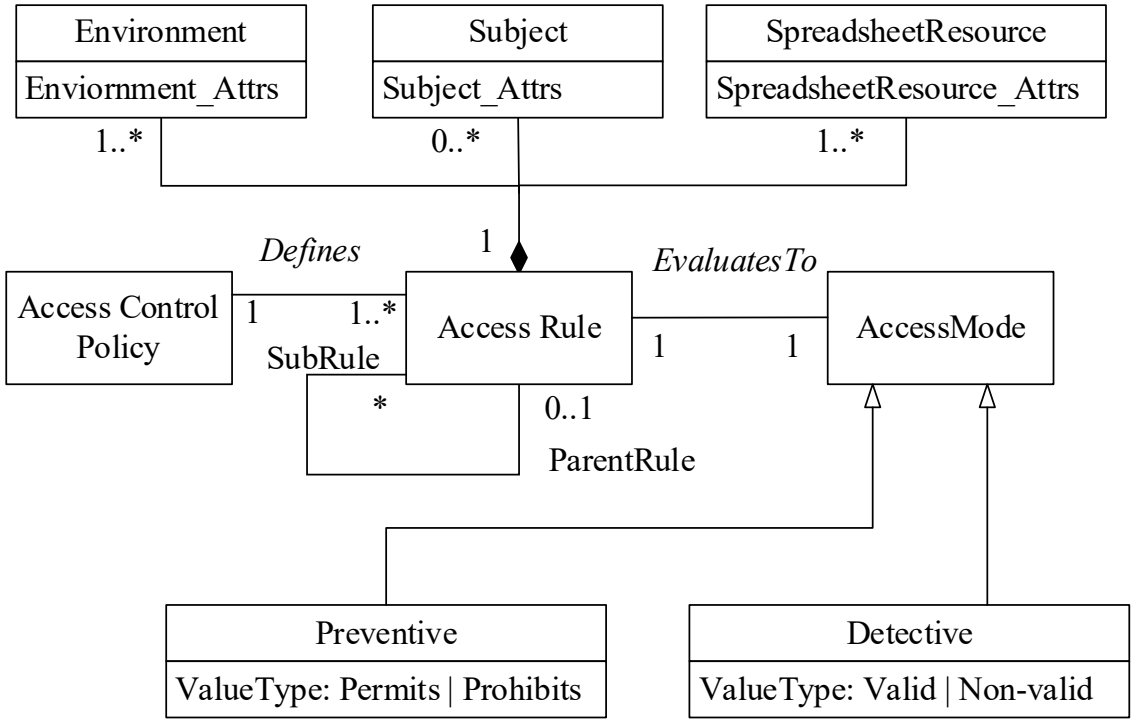

Figure 5. ABAC4S protocol access rules for spreadsheets.

- The class *AccessRule* represents the key entity in ABAC4S protocol access rules. Each access rule instantiated from *AccessRule* class is composed of associated spreadsheet resource(s), subject(s) to which the access rule applies, and environment attributes that allow dynamic environmental conditions with ABAC4S access rules. For example, with environment attributes, ABAC4S protocol allows specification of different access rules for development, testing and production environments of spreadsheet programs. Other examples of environmental conditions include time, location, threat level, or temperature [6].
- The class *AccessMode* represents mode of operations for ABAC4S protocol. Practically, this means that ABAC4S protocol can be deployed in real use case scenarios as preventive or detective protocol. In preventive use case implementation, ABAC4S evaluates each user action and spreadsheet resource change before the action is recorded in spreadsheet application. As a result of access mode resolution in preventive mode of operation,

ABAC4S protocol proactively permits or prohibits spreadsheet state changes. In detective use case implementation, ABAC4S evaluates each user's action and spreadsheet state transition after the change has been recorded in spreadsheet. As a result of access mode resolution in detective mode of operation, ABAC4S protocol can only log and determine for each resource state transition validity of change according to access rules. These modes of operations are modeled in Figure 5. with *Preventive* and *Detective* classes. In general, preventive modes of operations are preferable in information systems, however practical implementation of preventive mode of operations is far more complex than detective mode of operation.

- The class *SpreadsheetResource* represents link to associated set of spreadsheet resource attributes. Simple and consistent naming convention has been utilized during development of ABAC4S protocol, where set of attributes are represented with suffix *_Attrs* concatenated to Class name that represents spreadsheet resource. For example, a set of attributes associated with class *Worksheet* is represented in model as *Worksheet_Attrs*. Depending on nature and number of modeled spreadsheet resource, corresponding attributes are represented with enumerated lists or key-value HashMap. For example, attribute type for *InputData* is represented with enumerated list *[Boolean, Integer, Number, String, Date, Array]*. Attribute name for *Worksheet* is represented with key-value dictionary *{"Worksheet.name":"First_Sheet"}* [48].

4.3.3. ABAC4S Protocol Algebraic Representation

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.

4.3.4. Spreadsheet Resources as Direct graph

Direct graph representation of spreadsheet resources effectively defines hierarchical nature of spreadsheet resources. Each spreadsheet resource is modeled as a node in direct graph representation of spreadsheets. Edges modeled as directed arrows represents hierarchical "has-a" relationships between spreadsheet resources. This type of relationship between spreadsheet resources implies that hierarchically upper (parent) spreadsheet resource, "owns" or "contain" other hierarchically lower (child) spreadsheet resources. Visually, spreadsheet resource SR as a graph node is presented in Figure 6.

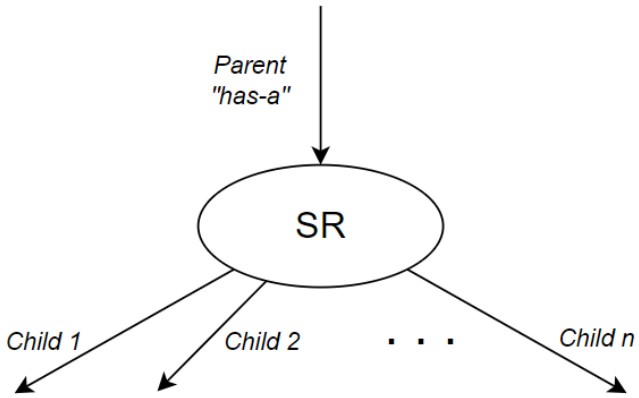

Figure 6. Spreadsheet resource graph node.

The following are definitions of important properties for directed graph representation of spreadsheet resources.

Definition 1 (Single Parent Property):

Each spreadsheet resource represented as a node in direct graph representation of the spreadsheet resources has one (and only one) parent node.

□

Definition 2 (Root Node):

Spreadsheet node is the root parent node in direct graph representation of spreadsheet resources, without parent node. All other spreadsheet resources are hierarchically lower nodes or child nodes related to spreadsheet root parent node.

□

Definition 3 (Single Root Path):

Each spreadsheet resource represented as a node in direct graph representation of the spreadsheet resources has one (and only one) unique path to spreadsheet root parent node.

□

Visually, single root path property is presented in Figure 7.

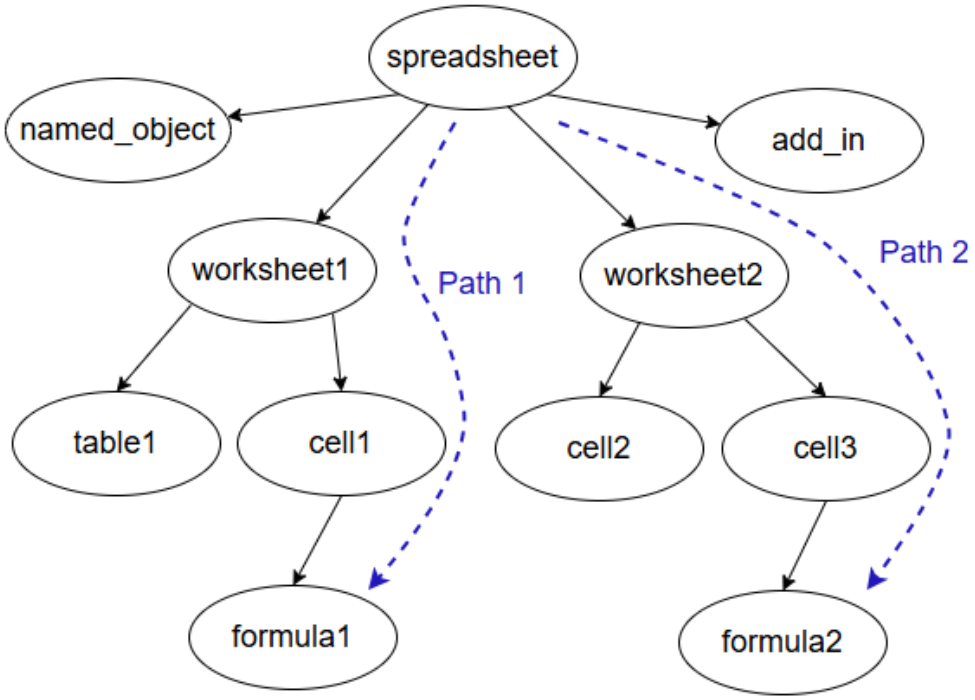

Figure 7. Single root path property for spreadsheet resources.

Example of spreadsheet representation with directed graph in Figure 7. has two distinguished paths marked for *formula1* and *formula2* resources. *Path1* is a unique single root path for formula1 and is determined with the following nodes:

$$spreadsheet \rightarrow worksheet1 \rightarrow cell1 \rightarrow formula1. \quad (6)$$

Path2 is a unique single root path for formula2 and is determined with the following nodes:

$$spreadsheet \rightarrow worksheet2 \rightarrow cell3 \rightarrow formula2. \quad (7)$$

4.4. ABAC4S Protocol Sequence Diagrams

A sequence diagram, in the context of presented ABAC4S protocol for automated quality assurance of spreadsheets is a visual representation of the interaction between entities in a system, focusing on the order and timing of messages exchanged. Specifically, a sequence diagram provides a visual timeline of events within a system, making it easier to understand the

sequence of actions and messages exchanged between protocol entities. Sequence diagrams used in this thesis are type of interaction diagram within Unified Modeling Language (UML) specification. As already stipulated in conceptual model of rules, ABAC4S protocol can be implemented as preventive or detective protocol. In preventive use case implementation, ABAC4S evaluates each user action and spreadsheet resource change before the action is recorded in spreadsheet application. Preventive mode of ABAC4S protocol implementation is presented in Figure 8.

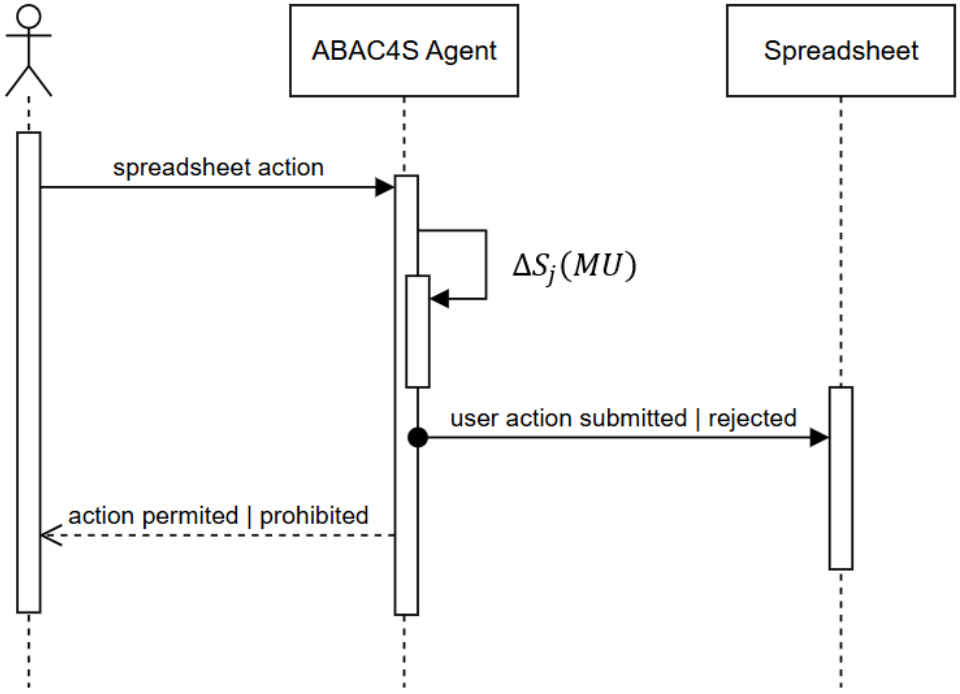

Figure 8. Sequence diagram for preventive mode of ABAC4S protocol implementation.

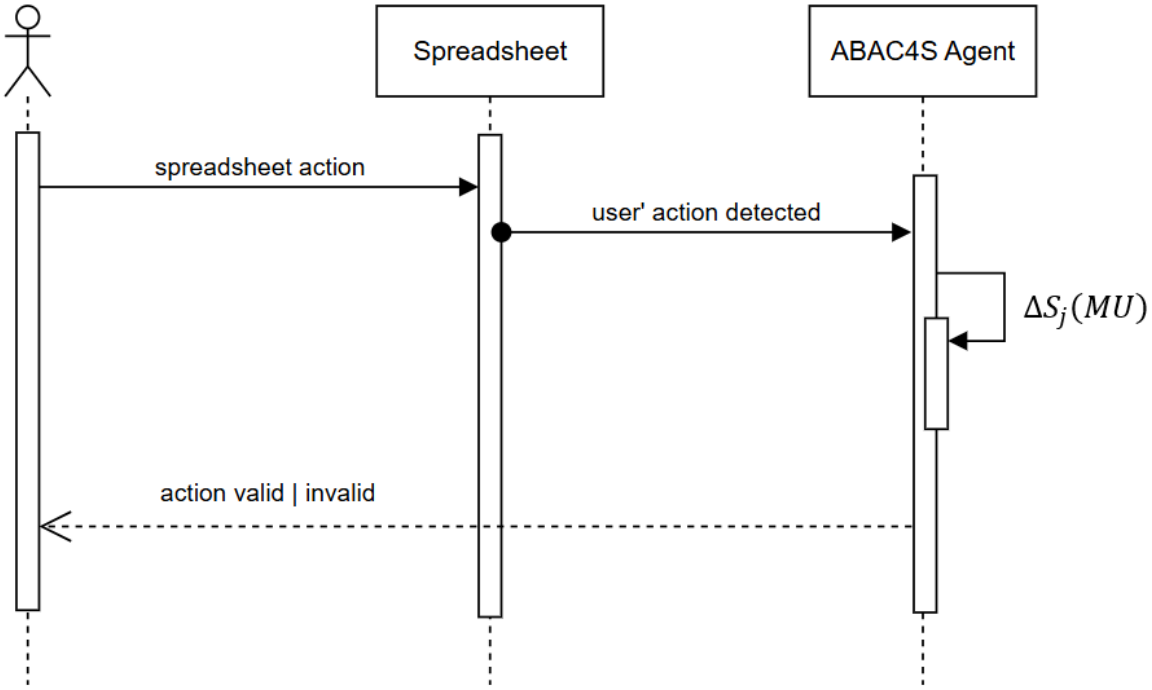

Figure 9. Sequence diagram for detective mode of ABAC4S protocol implementation.

In detective use case implementation, ABAC4S evaluates each user's action and spreadsheet state transition after the change has been recorded in spreadsheet. Preventive mode of ABAC4S protocol implementation is presented in Figure 9. In the above sequence diagrams, ABAC4S protocol computational logic is presented with "ABAC4S Agent". The goal is to demonstrate that ABAC4S computational logic is independent and agnostic to commercial spreadsheet applications. In practical deployment scenarios for ABAC4S protocol, "ABAC4S Agent" could be implemented as add-in to commercial spreadsheets (such as Microsoft Excel or LibreOffice Calc) installed on user's personal computer or microservice for cloud-based spreadsheets (such as Microsoft Excel Online or Google Sheets).

4.5. ABAC4S Protocol Access Rules Encoding

As already stipulated in ABAC4S protocol algebraic representation, access rules are modelled as quadruples (4), where A is a set of actions that user might perform on spreadsheet resource represented with enumerated list usually denoted with CRUD acronym (5). In addition to controlling user's action with access rules, ABAC4S protocol is designed to control structural properties of spreadsheet resources SR. The following are structural rules defined for spreadsheet resources.

Structural Rule 1: "Dot" notation is used to access specific resource attribute.

- Definition : *Resource.attribute*
- Example: `Spreadsheet.name`

Structural Rule 2: Chaining "Dot" notation is used to access specific resource attribute within hierarchy of spreadsheet resources.

- Definition : *Resource_1.Resource_2…Resource_n.attribute*
- Example: `Spreadsheet1.Worksheet1.Input_Cell1.name`

Structural Rule 3: Each spreadsheet resource is uniquely identified and addressed by its name. Cells are uniquely identified and addressee either by name (assigned with Named Manger functionality) or by unique address in A1 Reference style or R1C1 reference style. Standard range notation with ":" operator is used to address ranges.

- Definition : *Resource_name_1.Resource_name_1…Resource_name_n.attribute*
- Example: `Spreadsheet1.Worksheet1.A1.name`

Structural Rule 4: Structured table references are used to identify spreadsheet tables and access whole tables, specific columns or intersection of columns and rows [50]. Complete definition of structured tables references syntax is provided in Appendix A.

Structural Rule 5: "TYPE" function call for resource attribute determines attribute data type.

- Definition : *TYPE(Resource.attribute)*
- Example: `TYPE(Cell.value)`

Structural Rule 6: Equal sign operator "=" evaluates resource attribute to specific value

- Definition : *Resource.attribute="attribute_value"*
- Example: `Cell.value = "3,14"`

Structural Rule 7: Optional values with one selected value out of predefined list of values are specified with Pipe "|" operator.

- Definition : *Resource.attribute=["Value 1" | "Value 2" | … | "Value n"]*
- Example: `Cell.value = ["Banana" | "Apple" | "Orange"]`

Structural Rule 8: Predefined data types are *Boolean, Integer, Number, String, Date, Tuple, List, Set, Dictionary, Formula, NmedRange, Error.*

- Definition : *TYPE(Resource.attribute)=[“Boolean” | “Integer” | “Number” | “String” | “Date” | “Tuple” | “List” | “Set” | “Dictionary” | “Formula” | “NamedRange” | “Error”]*
- Example: `TYPE(Cell.value)=”Boolean”`

It is important to note that error types might differentiate significantly between commercial spreadsheet applications. In the latest version of Microsoft Excel spreadsheet, error types can be evaluated with built-in excel function `ERROR.TYPE()`. This function returns a number corresponding to one of the error values in Microsoft Excel or returns the `#N/A` error if no error exists [49]. In typical implementation, `ERROR.TYPE()` is used in combination with `IF()` function to test for an error value and return a text string, such as a message, instead of the error value. `ERROR.TYPE()` function is called with following syntax:

```
= ERROR.TYPE(Error_val)
```

Where *Error_val* is required parameter whose identifying number will be determined according to Table 2. presented below. Although *Error_val* can be the actual error value, in typical implementation this will be a reference to a cell containing a formula or formula itself passed as nested parameter to function.

Table 2. Microsoft Excel errors and `ERROR.TYPES()` returned values.

Error_val	ERROR.TYPE returns
#NULL!	1
#DIV/0!	2
#VALUE!	3
#REF!	4
#NAME?	5
#NUM!	6
#N/A	7
#GETTING_DATA	8
Anything else	#N/A

Structural Rule 9: For *Integer, Number,* and *Date* data types "less than" operator "<" checks if resource attribute value is less than value specified on right side of operator.

- Definition : *Resource.attribute* < *"Less_than_value"*
- Example: `Cell.value < "3,14"`

Structural Rule 10: For *Integer, Number,* and *Date* data types "greater than" operator ">" checks if resource attribute value is greater than value specified on right side of operator.

- Definition : *Resource.attribute* > *"Greater_than_value"*
- Example: `Cell.value > "3,14"`

Structural Rule 11: Logical AND operation in functional form might be used for evaluation of multiple criteria for identical spreadsheet resource. Multiple criteria should be separated with comma "," sign.

- Definition : *AND(Criteria1, Criteria2, …, Criteria2)*
- Example: `AND(Cell.value > "0", Cell.name="Sallary")`

Structural Rule 12: Logical OR operation in functional form might be used for evaluation of multiple criteria for identical spreadsheet resource. Multiple criteria should be separated with comma "," sign.

- Definition : *OR(Criteria1, Criteria2, …, Criteria2)*
- Example: `OR(TYPE(Cell.value)="String", Cell.name="Sallary")`

Structural Rule 13: Pattern matching for character combinations in strings is supported with regular expressions. "REGEX" function call with resource attribute passed as parameter checks for pattern matching. Regular expression patterns are provided on the right side of expressions.

- Definition : *REGEX(Resource)="regex_pattern"*
- Example: `REGEX(Cell1.value)="^\d{5}(?:[-\s]\d{4})?$"`

In the above example of pattern matching, regex pattern for US zip code is used to check if `Cell1.value` contains valid US zip code.

Structural Rule 14: Operations in functional form might be nested and composed with unlimited complexity and nesting depth. The only reasonable limitation is readability of final structural rule.

- Definition : *[OR | AND]([OR | AND]([OR | AND](Criteria_1, …, Criteria_n)))*
- Example: `AND(TYPE(Cell.value)= "String",`
  `OR(Cell.value="Banana", Cell.value="Apple"))`

4.6. ABAC4S Protocol Processing Logic

The Processing Logic of ABAC4S protocol for automated quality assurance of spreadsheets controls evolution of spreadsheet programs according to ABAC4S protocol access rules and structured criteria defined for spreadsheet resources. ABAC4S protocol controls two quality assurance criteria defined with protocol access rules; first behavioral criteria specified with user's roles and actions, and second structural properties of spreadsheet programs with set of structural rules specified on granular level of spreadsheet resources that constitute spreadsheet program. ABAC4S processing logic is divided into three consecutive processing steps that are executed for each spreadsheet state transition during spreadsheet lifecycle stages. During the first step of ABAC4S protocol processing logic, direct graph hierarchical representation of spreadsheet resources is generated. During the second step, resolution of potential conflicts for graph representation of spreadsheet resources and defined access rules is determined to ensure correct ABAC4S processing. During the last and third step of ABAC4S protocol processing logic, direct graphs representing two consecutive spreadsheet states are compared to determine transition $\Delta S_j(MU)$ between spreadsheet states S_j and S_{j+1} during spreadsheet lifecycle. Based on all identified modifications to affected spreadsheet resources, ABAC4S protocol compares

actions and spreadsheet resource modifications with defined access rules and determines validity of spreadsheet transition between corresponding states. In continuation of this thesis, comprehensive description for each ABAC4S protocol processing step is defined.

4.6.1. Generating Direct Graphs for Spreadsheet States

During the first step of ABAC4S protocol processing logic, for each spreadsheet lifecycle state S_j and S_{j+1}, direct graph with hierarchical representation of spreadsheet resources is generated according to defined rules for spreadsheet representation. Specifically, each generated directed graph must comply with Definition 1 (Single Parent Property), Definition 2 (Root Node) and Definition 3 (Single Root Path) introduced in Chapter 4.3.4 of this thesis.
To demonstrate generation of direct graphs for two spreadsheet states, simple spreadsheet example will be used.

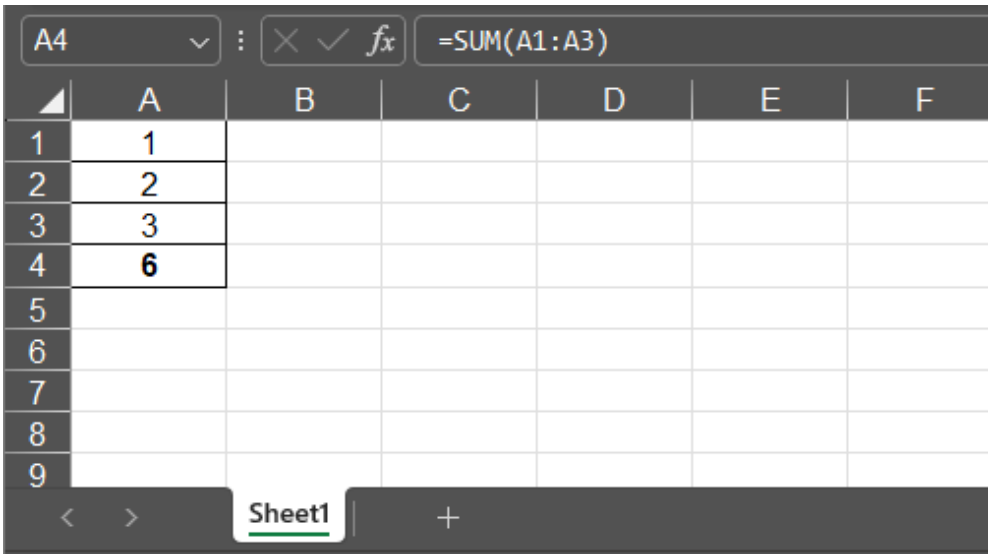

Figure 10. Example of spreadsheet graph at state S_j

Following spreadsheet resources constitute spreadsheet program "graph.xlsx" at state S_j represented in Figure 10.:

- Worksheet "Sheet1": Cell A1 contains value "1".
- Worksheet "Sheet1": Cell A2 contains value "2".
- Worksheet "Sheet1": Cell A3 contains value "3".
- Worksheet "Sheet1": Cell A4 contains formula: `=SUM(A1:A3)`, which evaluates to value "6".

Direct graph generated from spreadsheet program *"graph.xlsx"* at state S_j is visually represented in Figure 11.

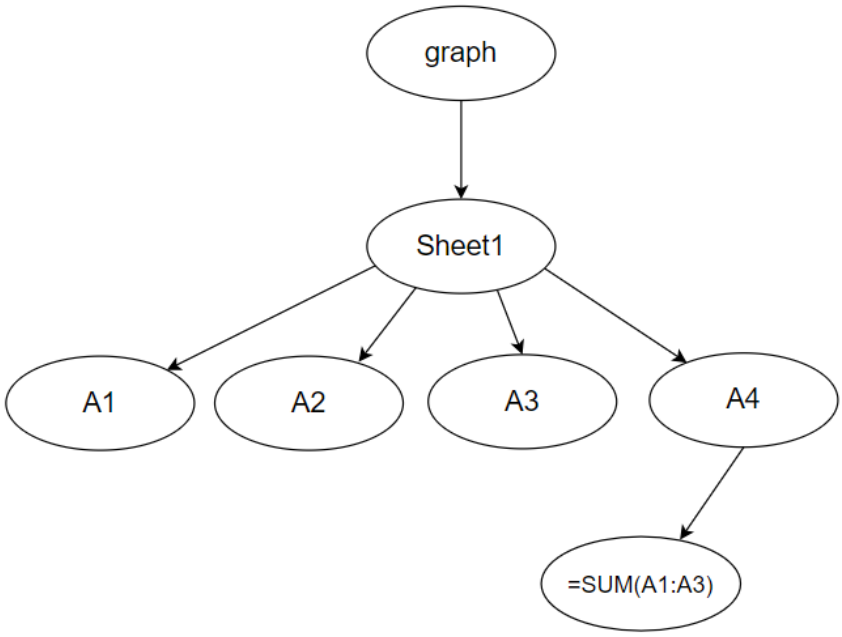

Figure 11. Directed graph representation of spreadsheet at state S_j

After modifications to spreadsheet resources, the spreadsheet state transition to new state S_{j+1} is visually represented in Figure 12.

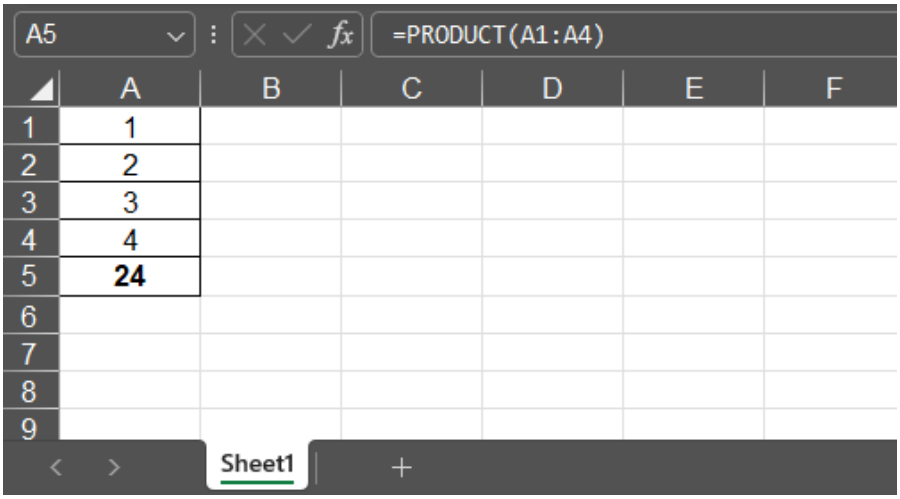

Figure 12. Example of spreadsheet graph at state S_{j+1}

Following spreadsheet resources constitute spreadsheet program "graph.xlsx" at state S_{j+1} represented in Figure 12.:

- Worksheet "Sheet1": Cell A1 contains value "1".
- Worksheet "Sheet1": Cell A2 contains value "2".

- Worksheet “Sheet1”: Cell A3 contains value “3”.
- Worksheet “Sheet1”: Cell A4 contains value “4”.
- Worksheet “Sheet1”: Cell A5 contains formula: `=PRODUCT(A1:A4)`, which evaluates to value “18”.

Direct graph generated from spreadsheet program *“graph.xlsx”* at state S_{j+1} is visually represented in Figure 13.

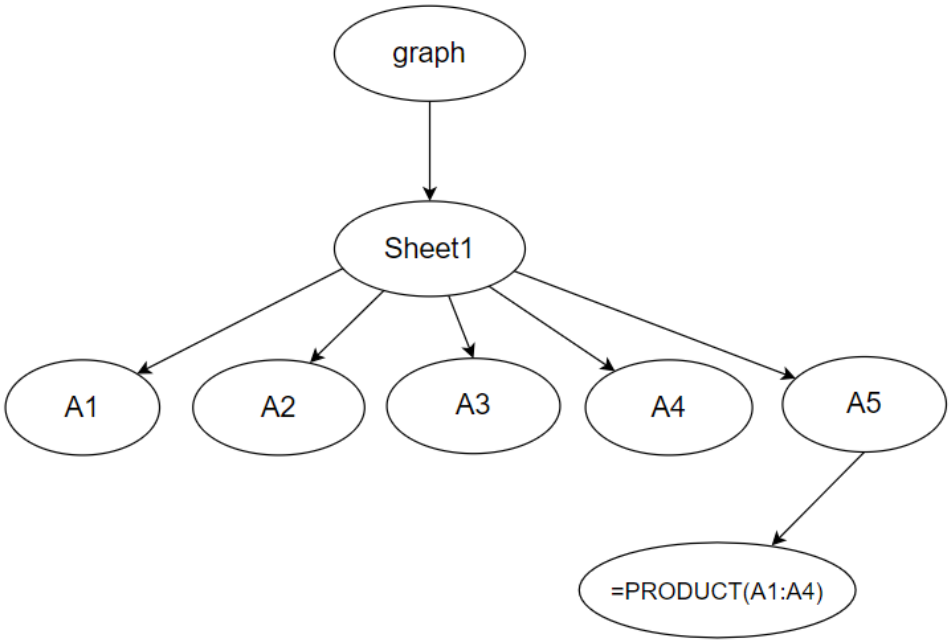

Figure 13. Directed graph representation of spreadsheet at state S_{j+1}

4.6.2. Conflict Resolution for Access Rules

During the second step, resolution of potential conflicts for graph representation of spreadsheet resources and defined access rules is determined to ensure correct ABAC4S processing. ABAC4S protocol access rules are defined as a set of quadruples. There is no limitation in the number of quadruples created to model specific user role. This results with great flexibility and granularity for ABAC4S access rules, because each spreadsheet resource behavioral and structural properties can be tightly controlled during the whole lifecycle. The negative side of this flexibility are potential conflicts in resolution of effective user roles and consequently inability of ABAC4S protocol to determine validity of spreadsheet transition between two states. The following definitions for ABAC4S protocol processing logic are introduced to resolve potential conflicts detected for user’s access rules.

Definition 4 (Priority of Actions):

Actions assigned to users are evaluated in the following order:

$$DELETE > CREATE > UPDATE > READ. \quad (8)$$

□

Definition 5 (Access Rule Inheritance):

All spreadsheet's resources inherit access rules applicable to their parents. To break chain of inheritance for spreadsheet resource, its parent should not have explicit access rule assigned ("Deny All Property" if explicit access rule for spreadsheet resource is not assigned).

□

Definition 6 (Spreadsheet Valid State):

Spreadsheet is in valid state S_{j+1}, iff Definition 4 (Priority of actions) and Definition 5 (Access Rule Inheritance) are satisfied for all affected spreadsheet resources during spreadsheet state transition from S_j to S_{j+1}.

□

For example, let's say that user `analyst` has two access rules assigned to control spreadsheet program "graph.xlsx" represented in Figure 10.:

```
(analyst, update, Sheet1, Instance = ”graph.xlsx”)∧
(analyst, delete, Sheet1.A4, Instance = ”graph.xlsx”)
```

Above access rules specify role for user `analyst` that allows the user to update worksheet Sheet1 and delete cell A4 within Sheet1. This combination of access rules creates conflict in resolution, because actions allowed for cell A4 have higher order than worksheet Sheet1 that is parent to cell A4. During conflict resolution, all spreadsheet resources inherit actions from their parent (Definition 5 for Access Rule Inheritance) and always actions with higher priority are enforced (Definition 4 for Priority of Actions). Important to note is that Definition 1 (Single Parent Property) and Definition 3 (Single Root Path) guarantee that for each spreadsheet program represented as hierarchical directed graph of spreadsheet resources only one path from each spreadsheet resource to root node exists. For simple spreadsheet program "graph.xlsx" root paths for all spreadsheet resources are following:

$$graph \rightarrow Sheet1 \rightarrow A4 \rightarrow SUM(A1{:}A3). \quad (9)$$

$$graph \rightarrow Sheet1 \rightarrow A4. \quad (10)$$

$$graph \rightarrow Sheet1 \rightarrow A3. \quad (11)$$

$$graph \rightarrow Sheet1 \rightarrow A2. \quad (12)$$

$$graph \rightarrow Sheet1 \rightarrow A1. \quad (13)$$

$$graph \rightarrow Sheet1 \quad (14)$$

Intuitively, conflict resolution is easier to understand if we look first what action are allowed for "Parent" resource, and naturally with inheritance applied the same action should be applied to its "children". In the above practical example, conflict resolution algorithm will enforce actions allowed on Sheet1 (update) to cell A5 and will forbid deletion of cell A4 within Sheet1 worksheet for user analyst.

Based on introduced definitions for resolution of conflict detected within specified access rules and given hierarchical graph representation of resources at spreadsheet state S_j, following algorithm `ResolveConflict` is defined in following Table 3.

Table 3. Algorithm `ResolveConfig`

Algorithm 1 *ResolveConflict(DG: SpreadsheetGraph, AR: AccessRules)*

```
DG: EDG ← SpreadsheetGraph
ForEach Node in EDG:
    ForEach Rule in AccessRules:
        If Node[Name] == Rule[SpreadsheetResource]
            Node[Action] ← Rule[Action]
ForEach Node in EDG:
    If Node[Action] <> Predecessor(Node[Action])
        Node[Action] ← Predecessor(Node[Action])
return EDG
```

Algorithm `ResolveConflict` in above pseudo code resembles elements of syntax utilized in Python programming language [53] and NetworkX graph processing library [54]. It is presented as a function that takes generated `SpreadsheetGraph` and `AccessRules` as parameters. Algorithm is structured around two traversals through EDG object (short for

Effective Direct Graph) instantiated at the beginning from passed `SpreadsheetGraph` object of type DG (short for Directed Graph). Within nested loop of the first pass through graph EDG, for matched graph node and spreadsheet resource in access rule, action attribute of the node is set to action attribute of matching access rule. During the second pass through graph EDG, action attribute of each node is compared with action attribute of its predecessor node. In case of differences identified, action attribute of the node is set to action attribute of its predecessor node. The function `Predecessor()` can be simply determined due to the Definition 1 for Single Parent Property of the graph representation for spreadsheet resources. In case multiple access rules tuples are passed as parameter, the last tuple determined during iteration through collection of all access rules with matching spreadsheet resource attribute will be used to assign action attribute to generated EDG graph. This is intentional behavior as multiple access rules for the same spreadsheet resource and the same user creates contradiction. In case such cases are identified during processing access rules, only the last definition is processed as valid and relevant while all others are ignored. As a result of algorithm processing generated EDG graph with resolved potential conflicts in access rules is returned. Action attributes for each node in returned EDG graph contain effective access rules determined., without potential conflicts in resolution.

4.6.3. Determination of Changes Between Two Spreadsheet States

During the last and third step of ABAC4S protocol processing logic, direct graphs representing two consecutive spreadsheet states are compared to determine transition $\Delta S_j(MU)$ between two spreadsheet lifecycle states. Transition $\Delta S_j(MU)$ between spreadsheet states S_j and S_{j+1} is defined as union of all modifications M performed by user U on affected spreadsheet resources. Based on all identified modifications to affected spreadsheet resources, ABAC4S protocol evaluates transition $\Delta S_j(MU)$ and determines its validity by comparing performed spreadsheet resources modifications with defined ABAC4S protocol access rules.

$$ABAC4S\big(\Delta S_j(MU), AccessRules\big) = \begin{cases} Permited \mid Prohibited;\ (preventive\ mode) \\ \\ Valid \mid Invalid;\ (detective\ mode) \end{cases} \quad (15)$$

In preventive implementation, ABAC4S protocol evaluates transition $\Delta S_j(MU)$ and user's access rules before the modifications are submitted to spreadsheet application and determines

if actions are permitted or prohibited. In detective implementation, ABAC4S protocol evaluates transition $\Delta S_j(MU)$ and user's access rules after the change(s) have been recorded in spreadsheet applications and determines if performed actions are valid or invalid.

With spreadsheet resources represented as directed graphs, determination of transition $\Delta S_j(MU)$ between two spreadsheet lifecycle stages becomes a challenge. This evaluation between two graphs is based on matching and comparing vertices and edges of the two involved graphs. The graph matching methods can be divided into two broad categories: exact graph matching and error-tolerant graph matching. Exact graph matching addresses the problem of detecting identical (sub)structures of two graphs g_1 and g_2 and their corresponding attributes [55]. In the context of ABAC4S protocol transition determination between two spreadsheets states, only exact graph matching algorithms are evaluated. Error-tolerant graph matching algorithms for spreadsheet state transitions are not evaluated in this research, as this would result in unpredictable determination of modifications performed in spreadsheets and consequently undesired performance of ABAC4S protocol. In the following, general attributed graph definition is provided [56].

Definition 7 (Attributed Graph):

An attributed graph (AG) is represented by quadruple $AG = (V, E, \mu, \omega)$, such that:

- V is a set of vertices (nodes).
- E is a set of edges such as $E \subseteq V \times V$.
- $\mu: L \rightarrow L_V$ is a vertex labeling function which associates label l_V to vertex v_i.
- $\omega: E \rightarrow L_E$ is and edge labeling function which associates label l_E to edge e_i.
- L_V and L_E are vertex and edge attributes sets, respectively. These attributes can be given by a set of integers $L = \{1,2,3\}$, a vector space $L = \mathbb{R}^N$ and/or a finite set of symbolic attributes $L = \{x, y, z\}$, which can differ in their dimensions.

□

Definition 7 for attributed graph supports all requirements for direct graph representation of spreadsheet resources. For example, node attributes, such as Name, Action or Value associated with each node in spreadsheet graph representation can be handled with symbolic attributes set $L = \{Name, Action, \dots, Value,\}$, with dimensions suitable for each node.

Graph Edit Distance (GED) is a graph matching method used in graph theory to quantify the similarity or dissimilarity between two graphs. Analogous to string edit distance (like Levensthein string edit distance [57]), GED measures the minimum "cost" of transforming

graph g_1 into graph g_2 through a sequence of elementary graph edit operations on graph g_1. The allowed operations are inserting, deleting and/or substituting vertices and their corresponding edges. In the following, GED definition is provided [56].

Definition 8 (Graph Edit Distance):

Let $g_1 = (V_1, E_1, \mu_1, \omega_1)$ and $g_2 = (V_2, E_2, \mu_2, \omega_2)$ be two graphs. The graph edit distance (GED) between g_1 and g_2 is defined as:

$$GED(g_1,\ g_2) = \min_{e_1,\ldots,e_k \in \gamma(g_1,g_2)} \sum_{i=1}^{k} c(e_i) \qquad (16)$$

such that:

- c denotes the cost function measuring the strength $c(e_i)$ of an edit operation e_i.
- $\gamma(g_1,\ g_2)$ denotes the set of edit paths transforming g_1 into g_2.

□

The "edit operations" are atomic changes applied to the graph. Each operation is assigned a specific cost function $c(e_i)$. Cost function $c(e_i)$ can be uniform or weighted differently based on the specific application of GED algorithm. The "cost" of an edit path (sequence of k operations) $\sum_{i=1}^{k} c(e_i)$ is the sum of the costs of all individual operations within that path. The GED is then the minimum cost among all possible edit paths that transform graph g_1 into graph g_2.

Common elementary edit operations include:

- **Vertex Insertion** $(\emptyset \rightarrow \boldsymbol{v})$ **:** Adding a new vertex (node) to the graph.
- **Vertex Deletion** $(\boldsymbol{u} \rightarrow \emptyset)$ **:** Removing an existing vertex (node) from the graph.
- **Vertex Substitution** $(\boldsymbol{u} \rightarrow \boldsymbol{v})$**:** Changing the label or attributes of an existing vertex (node). This can also be interpreted as deleting a vertex (node) and inserting a new one with different attributes.
- **Edge Insertion**$(\emptyset \rightarrow (\boldsymbol{u}, \boldsymbol{v}))$**:** Adding a new edge between two existing vertices.
- **Edge Deletion** $((\boldsymbol{u}, \boldsymbol{v}) \rightarrow \emptyset)$**:** Removing an existing edge.
- **Edge Substitution** $((\boldsymbol{u}, \boldsymbol{v}) \rightarrow (\boldsymbol{u}', \boldsymbol{v}'))$**:** Changing the label or attributes of an existing edge.

Currently, the most efficient implementation of GED method is Depth-First Graph Edit Distance (DF-GED) algorithm [56]. In comparison with other well-known Astar GED Algorithm (A*GED) that is considered as a foundation work for solving GED [58], with DF-GED algorithm memory consumption is not exhausted.

ABAC4S protocol for automated quality assurance of spreadsheets utilizes GED method for determination of $\Delta S_j(MU)$ during spreadsheet state transition from S_j to S_{j+1}. Mapping between GED elementary operations and ABAC4S protocol action is provided in Table 4.

Table 4. Mapping between GED operations and ABAC4S actions

GED elementary operation	ABAC4S protocol action
Vertex Insertion $(\emptyset \rightarrow v)$	CREATE action. For example, the worksheet Dashboard is created in spreadsheet.
Vertex Deletion $(u \rightarrow \emptyset)$	DELETE action. For example, the worksheet Dashboard is deleted from spreadsheet.
Vertex Substitution $(u \rightarrow v)$	UPDATE action. For example, the worksheet Dashboard is renamed to Report.
Edge Insertion$(\emptyset \rightarrow (u, v))$	CREATE action to add new spreadsheet resource within hierarchy of other spreadsheet resources. This operation creates hierarchical "has-a" relationships between spreadsheet resources. For example, a cell A1 is inserted to the worksheet Dashboard.
Edge Deletion $((u, v) \rightarrow \emptyset)$	DELETE hierarchical relationship between spreadsheet resources. It is important to note that in the context of ABAC4S protocol, DELETE action for node (vertex) representing spreadsheet resource, automatically performs DELETE action for edge representing "has-a" relationship from its parent. For example, deletion of the worksheet Dashboard, removes spreadsheet resource Dashboard (node deletion) and its link to parent spreadsheet resource (edge deletion).
Edge Substitution $((u, v) \rightarrow (u', v'))$	UPDATE action to hierarchical relationship between spreadsheet resources. For example, the formula in cell A1 in worksheet Dashboard is moved from cell A1 in worksheet Dashboard to cell A1 in worksheet Report.

To demonstrate determination of $\Delta S_j(MU)$ during spreadsheet state transitions, spreadsheets represented in Figure 11. and Figure 13. will be analyzed with the GED method.

Spreadsheet resources that are affected as part of spreadsheet state transition are highlighted with light blue color in Figure 14.

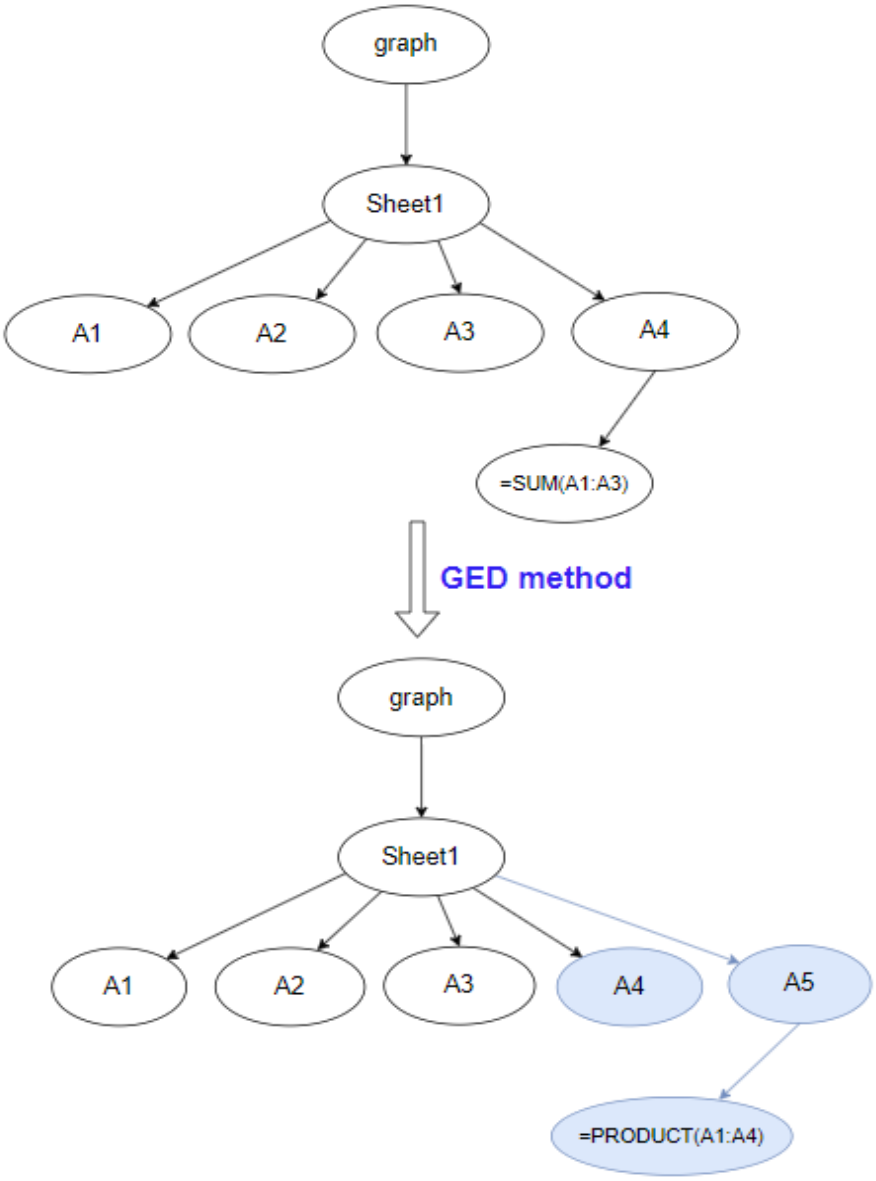

Figure 14. Visualized transition between two graphs

Mapping between GED elementary operations and ABAC4S protocol action for this example is provided in Table 5.

Table 5. Example of mapping between GED and ABAC4S actions

GED elementary operation	ABAC4S protocol action
Vertex Substitution ($Sheet1.A4.Value =$ "$=SUM(A1:A3)"$ → $Sheet1.A4.Value =$ "4")	UPDATE action. Cell A4 in worksheet Sheet1 is updated from value „=SUM(A1:A3) to new value „4“
Vertex Insertion ($\emptyset$ → $Sheet1.Cell.Address =$ "$A5$")	CREATE action. Cell A5 in worksheet Sheet1 is created.
Edge Insertion ($\emptyset$ → ($Sheet1, A5$))	CREATE action. Worksheet Sheet1 “has-a” new cell A5 (i.e., a cell A5 is inserted to the worksheet Sheet1).
Vertex Insertion ($\emptyset$ → $Formula.Value$ = "$=PRODUCT(A1:A4)"$)	CREATE action. New formula is created with value „=PRODUCT(A1:A4)“.
Edge Insertion ($\emptyset$ → ($Sheet1.A5, Formula.Value =$ "$=PRODUCT(A1:A4)"$))	CREATE action. Cell A5 within worksheet Sheet1 “has-a” formula new with value „=PRODUCT(A1:A4)“.

Under the assumption that cost for individual GED edit operation is 1 in the above example, the minimum total cost of the spreadsheet transition determined by GED algorithm is 5.

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:

$$M = (S, I, R, L). \tag{17}$$

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 $s \in S$ is a sequence of states $\pi = s_0, s_1, \dots, s_n$ 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:

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

CTL temporal operators are as follows:

- $\boldsymbol{X}\ \phi$ – ϕ holds ne**X**t time
- $\boldsymbol{F}\ \phi$ – ϕ holds sometime in the **F**uture
- $\boldsymbol{G}\ \phi$ – ϕ holds **G**lobally in the future

- $p\ \boldsymbol{U}\ \phi$ – ϕ holds **U**ntil ϕ holds

CTL allows modeling complex behavior of the systems, where temporal operators must always be preceded by a path quantifier. Figure 15. visually represents the meaning of CTL path and temporal operators (adapted from [31]).

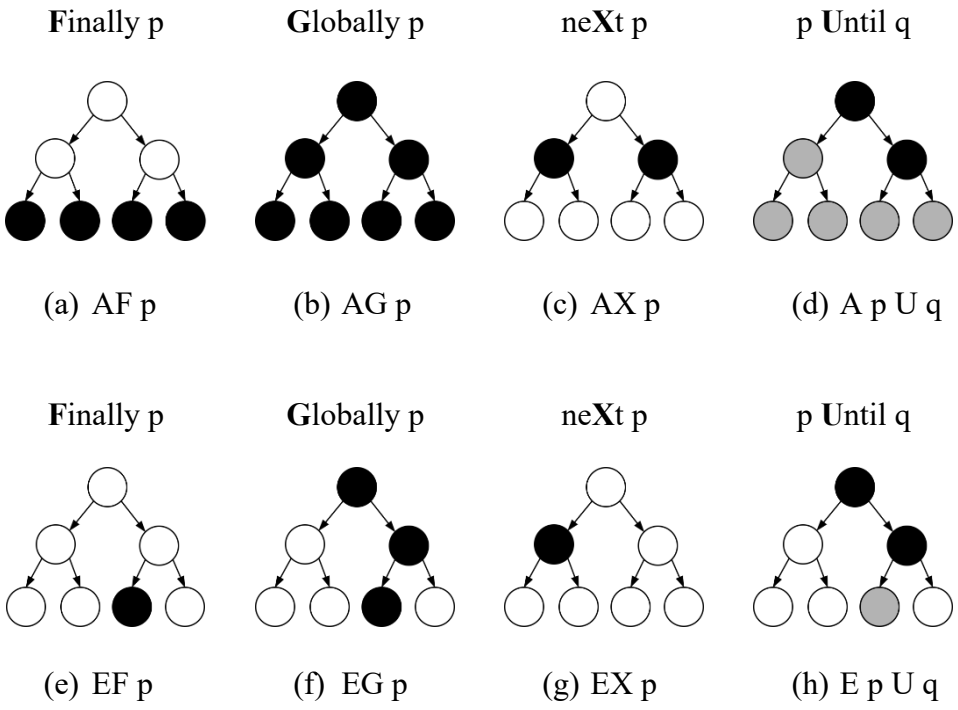

Figure 15. CTL path and temporal operators.

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 $M \models \phi$. Operator $\models$ meaning is "specification ϕ is satisfied by structure M".

6. Model Checking the ABAC4S Protocol

Model checking of the proposed ABAC4S protocol for spreadsheets has been performed 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 development of NuSMV model with the SMV language has been the abstraction of the provided spreadsheet conceptual model with suitable SMV module. The hierarchy of created SMV modules follows the natural hierarchy of spreadsheet resources defined in the spreadsheet conceptual model and presented visually in Figure 16.

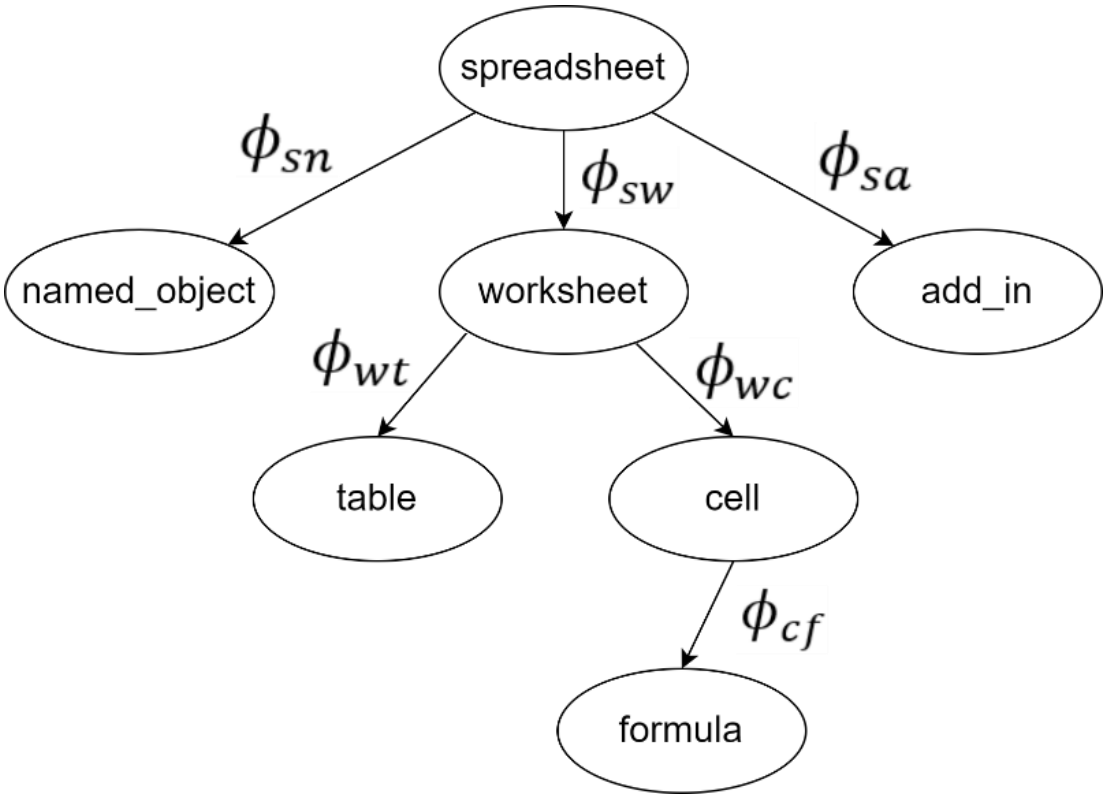

Figure 16. Hierarchy of spreadsheet resources as SMV language modules.

All possible access role combinations and defined CRUD user actions have been explored during research presented in this thesis. In such a scenario, correct protocol behavior should detect and resolve potential conflicts during the consecutive model state. Below is the hierarchical model of spreadsheet resources specified in SMV language.

```
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, capability of the SMV language to construct hierarchical modules that correspond to the natural hierarchy of spreadsheet resources have been utilized in developed SMV modules. 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, each spreadsheet resource has been abstracted and simplified to a bare minimum, without loss for ABAC4S protocol correctness. In case more complex representation is required with additional two attributes on module `spreadsheet_t()`, they can be added with an enumerated list of constants. Next case assignment for added attributes in SMV code specification are sharing the same structure with original 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 16., 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 ($\phi_{cf}, \phi_{wc}, \phi_{sw}$). Fragment of SMV code for ϕ_{sw} next-case conjunction statement that specifies logic for priority of actions and access inheritance protocol rules is listed below. The complete SMV source code for ABAC4S protocol specification is provided in Appendix B. of this thesis and author's GitHub repository [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, model checking has been conducted with the NuSMV model checker. NuSMV model checker has been utilized in interactive mode with support of NuSMV built-in shell for execution of CTL temporal logic property checks.

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

$$AG\ (p\ \rightarrow AF\ q). \qquad (18)$$

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. Spreadsheet Quality Assurance

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. According to the International Organization for Standardization (ISO), Quality Assurance (QA) is a systematic process that provides confidence that a product, service, or process meets quality requirements. QA plays an instrumental role in fostering a culture of constant, ongoing improvement. QA involves planned and systematic actions to achieve this confidence, often implemented within a quality management system. The primary aim of QA is to reduce the risk of defects – and importantly, to address faults as early as possible in the value chain [59]. In the context of spreadsheets QA, this means putting in place both technical and managerial processes and controls to ensure that quality attributes are fulfilled during whole lifecycle of spreadsheets.

7.1. Spreadsheet Quality Model

While early spreadsheet use was often informal, the growing reliance on them for critical decision-making process within enterprise triggered significant evolution in spreadsheet quality assurance practices. In one of the first attempts to formalize quality assurance principles, 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. As part of the research conducted, O'Beirne presented practical checks and control procedures to increase the quality of spreadsheet programs.

Further refinement in spreadsheet quality research provided a set of domain specific metrics, used to measure concrete spreadsheet characteristic [14]. Based on widely accepted ISO/IEC 9126 international standard for software product quality [15], Peixoto developed a model of quality for spreadsheets, defining all the features that are important on a spreadsheet and how the quality of that feature can be quantified [51]. The author provided a comprehensive analysis of ISO/IEC 9126 standard and mapped relevant quality attributes to spreadsheets. Spreadsheet quality model is visually presented in Figure 17. [51].

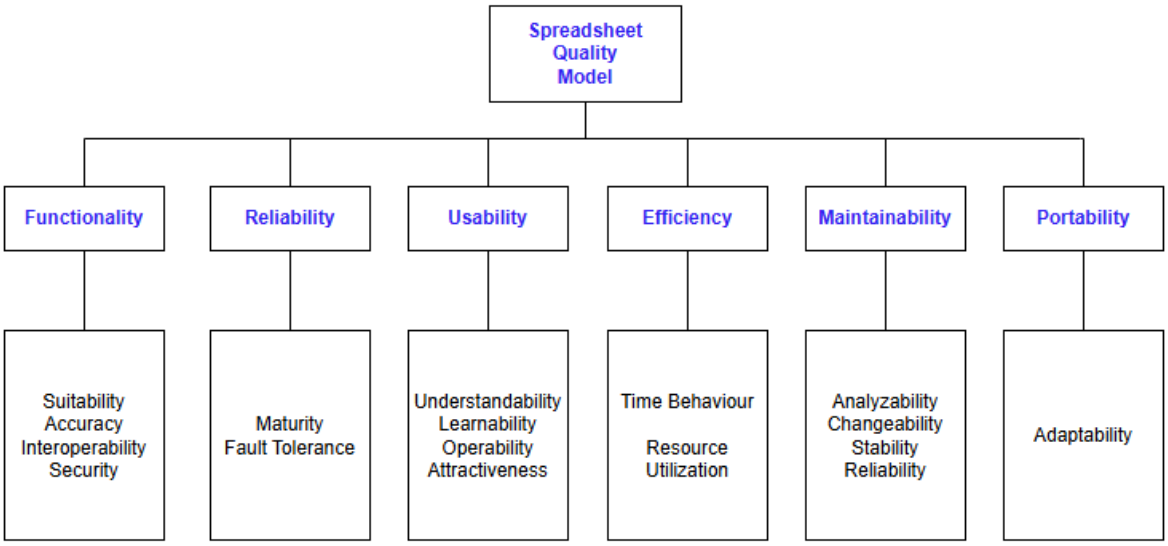

Figure 17. Spreadsheet quality model

In continuation of the research presented in this thesis, each of the characteristic defined in spreadsheet quality model is discussed in the context of novel ABAC4S protocol for automated quality assurance of spreadsheets.

7.1.1. Functionality

Functionality is the capacity of the spreadsheet to satisfy the user's needs, either implied or stated. It is divided into the following sub-categories:

- Suitability: Is the quality of having the properties that are right for a specific purpose.
  - Number of incongruences.
  - Number of references to blank cells in formulas.
- Accuracy: Is the faithful measurement or representation of the true, correctness.
  - Number of output cells with errors or invalid content.
- Interoperability: Is the ability of two or more spreadsheet resources to exchange information and to use the information that has been exchanged.
  - Data exchanged between worksheets.
  - Quantity of rightful formulas.
  - Total number of cells with references.
- Security: ensure confidentiality and integrity of spreadsheet programs and data.
  - Protection of raw data in cells.
  - Protection of data structures and data types.

- Use of data validation constraints.
- Ensure confidentiality and restricted access to users.

ABAC4S protocol can effectively control suitability, accuracy, interoperability and security of spreadsheet programs. With granular composition of ABAC4S access rules, structural characteristics for each spreadsheet resource can be controlled to prevent dissonances between initial user's expectations and actual solution developed, prevent references to empty formulas, enforce validation through access rules allowable types and data contained in cells. ABAC4S protocol brings additional controls to ensure confidentiality and integrity of spreadsheet programs and data that are not currently available in commercial spreadsheet applications. For example, password locks on workbooks or worksheets can enforce only discretionary access control where spreadsheet owners can control basic access rights to users. With granular implementation of access rights for different users, ABAC4S can enforce complex authorizations for spreadsheet programs required in multi-user environments.

7.1.2. Reliability

Reliability is the capacity to maintain its level of performance under stated conditions during the whole lifecycle of spreadsheet program. It is divided into the following sub-categories:

- Maturity: The state of quality of being fully developed and up to date with the latest user's requirements.
  - Number of empty labeled rows and columns.
  - Number of orphan worksheets, cells, formulas and other spreadsheet resources not used as part of spreadsheet calculations.
- Fault Tolerant: Quality characteristics to continue operating properly in the event of one or more faults within spreadsheet component.
  - Overall complexity and number of cells and other spreadsheet resources used to perform calculation.
  - Number of complex formulas.
  - Number of complex user-defined functions and other custom components.

ABAC4S protocol can effectively control maturity and fault tolerance of spreadsheet programs. Mature design and layout of spreadsheet resources within spreadsheets, complexity

of formula and user defined functions can be tightly controlled with ABAC4S access rules structural characteristics. In addition, behavioral aspects of ABAC4S access rules enables enforcement of spreadsheet development standards within multi-user environments and proper segregations of duties between developers, data inputters and other spreadsheet users. Complexity of externally built spreadsheet modules can be controlled with ABAC4S access rules through validation of input parameters and returned results.

7.1.3. Usability

Usability is the property of the spreadsheet to be understood by users, enabling seamless user experience. Usability is characterized by consistency of spreadsheet program, accessibility, and overall user's satisfaction. It is divided into the following sub-categories:

- Understandability: Spreadsheet property to be understood by users.
    - Intuitive colors, layout and structure of spreadsheet resources utilized by users during interaction with spreadsheets.
    - Separation between input, computation and output.
    - Total number of spreadsheet resources utilized for user's interaction.
- Learnability: Spreadsheet property to enable fast and fluent adoption of spreadsheet programs for new users.
    - Total number of spreadsheet resources used for interaction with users.
    - Color coding and layout style for spreadsheet resources used for interaction with users.
    - The amount and structure of the data flows between worksheets and other spreadsheet resources.
- Operability: Spreadsheet quality property driving positive user experience while minimizing the number of users actions and overall user's fatigue.
    - Use of effective data validation and drop-down lists.
    - Controlled size and complexity of validation lists.
    - Separation between input, computation and output worksheets and associated computational resources.
    - Natural flow of information and actions from up to down and from left to right during user's interaction with spreadsheets.
- Attractiveness: Spreadsheet visual and functional attractiveness that enables positive user experience.

- Visual layout of input fields and output results.
- Attractive color coding.
- Overall spreadsheet design compliant with organizational development standards.

ABAC4S protocol can effectively control understandability, learnability, operability and attractiveness of spreadsheet programs. This can be achieved in a fully automated way, once proper access rules to support the above quality characteristics are developed. For example, once the visual layout for input worksheets and cells is codified in ABAC4S access rules, they can be used as templates and ensure that organizational visual standards are consistently deployed to all spreadsheet programs. Flexibility and granularity of ABAC4S access rules can effectively support variety of visual standards, layouts and color coding for spreadsheet cells and other visual elements used in spreadsheet programs.

7.1.4. Efficiency

Efficiency property of spreadsheets relates to the amount of spreadsheet resources utilized to achieve desired goal. It is divided into the following sub-categories:

- Time Behavior: Overall duration of time required to complete task modelled with spreadsheet program.
  - The number and complexity of formulas and other spreadsheet resources required for complex calculations.
  - Number of repetitive actions that are causing time inefficiencies.
- Resource Utilization: Overall resource utilization (processing power, memory consumption, slow interfaces to other systems) required to complete task modelled with spreadsheet program.
  - Number and complexity of spreadsheet resources that are increasing processor and memory consumption.
  - Ineffective data interfaces to other systems.

ABAC4S protocol can effectively control time behavior and resources utilization of spreadsheet programs. For example, with environment attribute specified in ABAC4S access rules, characteristics of spreadsheet execution environment for cloud-based spreadsheets can be specified. Characteristics and modularity of spreadsheet resources can be enforced with

access rules to ensure that desired resource utilization and time behavior of spreadsheet program is achieved and maintained during the whole spreadsheet lifecycle. Type of input parameters for user defined functions as well as other computational modules can be controlled with access rules to prevent wrong data types at the input. Size and structure of input parameters can also be controlled with ABAC4S protocol access rules to prevent uncontrolled resource utilization.

7.1.5. Maintainability

Maintainability is the property of spreadsheets to expand functionality or correct errors. It is divided into the following sub-categories:

- Analyzability: Characteristics to be analyzed and rich conclusions within the shortest possible period.
    - Overall number and complexity of spreadsheet resources utilized to develop spreadsheet programs.
    - Self-description of spreadsheet program through comments and structured naming convention.
    - Use of named ranges and named objects with intuitive names instead of cryptic names and cell references.
- Changeability: Time and resources required to change spreadsheet program.
    - Overall number and complexity of spreadsheet resources utilized to develop spreadsheet programs.
    - Modularity and granularity of spreadsheet resources utilized to develop spreadsheet programs.
    - Clarity of interfaces, parameters and returned results for computational spreadsheet resources.
- Stability: Frequency and resources required to maintain, fix and patch spreadsheet programs to sustain desired functionality.
    - Overall number and complexity of spreadsheet resources utilized to develop spreadsheet programs.
    - Number and complexity of data transformations.
    - Assumptions and exception management during computation and data transformations.
- Testability: Time and resources required to test spreadsheet program.

- Overall number and complexity of spreadsheet resources utilized to develop spreadsheet programs.
- Modularity and granularity of spreadsheet resources utilized to develop spreadsheet programs.
- Clarity of interfaces, parameters and returned results for computational spreadsheet resources.
- Proper data transformations and type inferences.

ABAC4S protocol can effectively control analyzability, changeability, stability and testability quality characteristics of spreadsheet programs. With access rules structural characteristics for each spreadsheet resource overall complexity can be controlled with details for input parameters and output results of each computational module. For example, if modularity characteristic requires that each user defined function accepts only one input parameter this can be easily achieved with access rules. This will prevent the creation of large and bulky user defined functions with multiple input parameters and enforce usage of smaller user defined functions that perform simple and testable computation. Desired result of computation and better information flow can be achieved with proper composition of smaller user defined functions. Consequently, this good development practice enforced through ABAC4S access rule will increase overall analyzability, changeability, stability and testability of spreadsheet program.

7.1.6. Portability

Portability is the quality characteristic of spreadsheet programs to run effectively within different spreadsheet execution environments. It is divided into the following sub-categories:

- Adaptability: To what extent can the spreadsheet program adapt to environmental change.
  - The number and complexity of unique spreadsheet resources used depends on certain versions of spreadsheet application and/or programming language.
  - Nonstandard data formats used.

ABAC4S protocol can effectively control the adaptability of spreadsheet programs. Key design requirements for ABAC4S protocol are independence from commercial spreadsheet application and closed vendor data exchange and data storage formats. ABAC4S protocol

access rules are defined in an open and simple text-based format. In use cases for ABAC4S protocol implementation presented in this thesis, representation of developed access rules is provided in open JavaScript Object Notation (JSON) format. In addition, other portability characteristics can be easily enforced with access rules. For example, use of functions that are portable between different spreadsheet applications can be enforced with ABAC4S access rules to improve overall portability of spreadsheet program.

7.2. Automated Spreadsheet Quality Assurance

In the work of Jannach et al. [63], automated spreadsheet quality assurance approaches have been classified into finer-grained scheme. The classification provided by authors analyzed various automated spreadsheet quality assurance approaches in terms of their capabilities to serve both finding (detection) and avoiding (prevention) errors and quality issues in spreadsheet programs. The following automated quality assurance approaches are presented in the work of Jannach et al. [63]:

- Visualization-based approaches: These approaches provide the user with a visually enhanced representation of some aspects of the spreadsheet to help him or her understand the interrelationships and dependencies between cells or larger blocks of the spreadsheet. These visualizations help the user to quickly detect anomalies and irregularities in the spreadsheet.
- Static analysis & reports: These approaches are based on static code analysis and aim to point the developer to potentially problematic areas of the spreadsheet. Examples of techniques include “spreadsheet smells” or the detection of data clones but also the typical family of techniques found in commercial tools capable of reporting summaries about unreferenced cells.
- Testing-based techniques: The methods in this category aim to stimulate and support the developer to systematically test the spreadsheet application during or after construction. The supporting tools for example include mechanisms for test case management, the automated generation of test cases or analysis of the test coverage.
- Automated fault localization & repair: The approaches in this category rely on a computational analysis of possible causes of an error or unexpected behavior through code debugging and analysis tools. They rely on additional input by the developer such as test cases or statements about the correctness of individual cells. Modern approaches

for automated fault localization and repair are based on Large Language Models (LLM) specifically trained for spreadsheets capable of providing "repair" suggestions and syntax reconstruction [22].

- Model-driven development approaches: Methods in this category mainly adopt the idea of using (object-oriented) conceptual models as well as model-driven software techniques during development of spreadsheet programs. The typical advantages of such approaches include the introduction of additional layers of abstraction or the use of code-generation mechanisms.
- Design and maintenance support: The approaches in this category either help the spreadsheet developer to end up with better error-free designs or support him or her during spreadsheet construction. The mechanisms proposed in that context for example include automated refactoring tools and spreadsheet code suggestions provided by tools such as FLAME language model [22].

Overview of main categories for automated quality assurance of spreadsheets proposed by Jannach et al. [63] is presented in Table 6.

Table 6. Summary of spreadsheet QA approaches

Automated Spreadsheet QA	Finding Errors	Avoiding Errors
Visualization-base approaches	x	x
Static code analysis and reports	x	x
Testing approaches	x	
Automated fault localization and repair	x	
Model-driven development approaches		x
Design and maintenance support		x

ABAC4S protocol for automated quality assurance of spreadsheet programs uniquely addresses both methods in focus of quality assurance research, finding (detection) and avoiding (prevention) of spreadsheet errors and quality issues. In comparison to other automated quality assurance approaches presented by Jannach et al. [63], unique properties of proposed ABAC4S protocol can be summarized as follows:

- ABAC4S protocol for automated quality assurance of spreadsheet programs allows comprehensive control over user's interactions with spreadsheets in enterprise environments based on roles and job descriptions modeled with ABAC4S access rules.

- ABAC4S protocol can control behavioral and structural quality criteria for spreadsheets on granular level of spreadsheet resources and complex authorization schemes present in multi-user environments.
- ABAC4S protocol can be combined successfully with other spreadsheet quality assurance approaches presented in the work by Jannach et al. [63], depending on needs and maturity of organizations willing to improve quality of their spreadsheets.

8. ABAC4S Protocol Use Cases

A practical example of ABAC4S protocol for automated quality assurance of spreadsheets in multi-user environments will be demonstrated with two use cases. In the first case, ABAC4S protocol was implemented to manage the lifecycle of spreadsheet used as IT Administrator logbook. The author of this thesis developed this spreadsheet use case as request from small company IT department to maintain structured IT Administrator logbook. In the second case, ABAC4S protocol was implemented to manage calibration processes within analytical laboratory. The Calibration process for Negative Temperature Coefficient (NTC) probes have been supported with spreadsheet program to record calibration results and provide structured reports to laboratory personnel. In both cases, standalone versions of Microsoft Excel spreadsheet have been used (Version 2505 Build 16.0.18827.20102, 64-bit). ABAC4S protocol computational logic has been developed by the author of this thesis as collection of scripts in Python programming language [54]. It is important to note that used scripts are not production ready and have been used primarily as a proof-of-concept research tool. Scripts have been executed asynchronously to simulate ABAC4S computational logic, including parsing of excel spreadsheets xlsx files and parsing of access rules as textual comma separated files with tuples representing ABAC4S access rules. An implementation of Depth-First Graph Edit Distance (DF-GED) algorithm from Python NetworkX library (`optimal_edit_paths` function) has been used to determine changes performed by users in spreadsheets [55]. In both cases, ABAC4S protocol has been deployed in detective mode where user's action and spreadsheet state transitions have been evaluated with ABAC4S protocol after the changes have been recorded in spreadsheets. The goal was to minimize impact on spreadsheet users, and existing habits on how users interact with their spreadsheets.

8.1. IT Administrator Logbook

Process of maintaining servers and networking equipment within small company is documented in IT Administrator logbook. On a weekly basis, computer administrators perform critical sets of activities to upgrade and maintain all required servers owned by company. Computer administrators are divided into two groups, first managing Microsoft Windows based servers, and second managing Linux based servers. It is important that all critical maintenance and patching activities are initiated and completed during Friday's afternoon with minimal impact to business processes. During the same planned maintenance period, IT network administrators should as well conduct key activities on networking equipment. Servers and

networking equipment have dedicated (Internet Protocol) IP addresses that uniquely identifies each host on the company network. Each administrator should log in to the spreadsheet diary logbook short information about impacted asset during and status of completed activities. The company has an internal software development department with experienced spreadsheet developer and structured development standard in place. Based on the provided short process description in place for maintenance activities, following spreadsheet structural requirements and user's roles are defined:

- Spreadsheet developer should perform coding and development activities on separate development instance named "diary_logbook_dev" and after successful testing, this development instance should be merged to production instance named "diary_logbook_prod"

- Diary logbook spreadsheet should contain two worksheets. First worksheet named "Logbook", should contain Table named "Logtable" with following columns used to document activities performed. "Seq" is auto incremented column documenting sequence number of performed activity. Second column is named "Date" which document the date when maintenance have been performed. Third column in table is named "IP Address" containing unique network address of asset where activities have been performed. Forth column named "Status" should contain values "Passed", "In Progress", "Rejected" or "Failed". In case of "Failed" status, backup team should work jointly with IT administrators to recover affected IT asset to latest possible state. State "Rejected" designates unsuccessful maintenance or patching activity, but affected asset is still operating with previous version of system software. In last column of the table, named "Group", IT Administrator group should be entered as "Windows", "Linux" or "Network", depending on IT Administrator responsibilities. Only developer is allowed to change worksheet "Logbook" structure and layout. IT Administrators should be allowed only to add new entries in the table "Logtable" according to limitations derived from their job responsibilities. IP addresses of affected assets should be controlled by access rules, and for example network administrator is not allowed to document activities for Microsoft Windows or Linux servers. Real IP addresses are obfuscated in this example with artificially created class C IP addresses for each relevant group of IT assets. Class C IP addresses are part of the IPv4 addressing scheme and are designed for

smaller networks used in home and private networks. Just for the purpose of this example, imaginary set of IP addresses for each IT asset group are defined as follows:

- Windows Servers IP address range: 192.168.1.10-192.168.1.100.
- Linux Servers IP address range: 192.168.1.101-192.168.1.200.
- Networking devices IP address range: 192.168.1.201-192.168.1.300.

- In second worksheet named “Dashboard”, aggregated statistics of all maintenance activities for certain date should be presented. Statistics should be presented in simple tabular format. Table named “MaintenanceStatus” should contain all data aggregated from worksheet “Logbook” with following columns. In first column named “Date”, summary of all activities from worksheet “Logbook” on certain date should be presented. In consecutive columns, four columns named “Passed”, “In progress”, “Rejected” and “Failed” are repeated for three different asset categories “Windows”, “Linux” and “Network”, thus providing aggregated view for three different groups of assets and IT administrators conducting maintenance activities. Only spreadsheet developer is allowed to change and modify this worksheet on developer instance. On production instance, both developer and all IT administrators should have view only authorizations. Color coding and visual structure of Dashboard should follow existing company’s software development standards in place.

With the above description of the maintenance process and spreadsheet structural requirements, developer access rules for ABAC4S protocol should be structured as follows:

```
(developer, create, Worksheet.name=”Logbook”,
Instance=”diary_logbook_dev”)∧
(developer, create, Worksheet.name=”Dashboard”,
Instance=”diary_logbook_dev”)∧
(developer, create, Logbook.Table.name=”Logtable”,
Instance=”diary_logbook_dev”)∧
(developer, create, Dashboard.Table.name=” MaintenanceStatus”,
Instance=”diary_logbook_dev”)∧
(developer, create, Logtable[#Headers]=[”Seq”,”Date”,”IP
Address”,”Status”,”Group”], Instance=”diary_logbook_dev”)∧
```

```
(developer, create, MaintenanceStatus[#Headers] =[”Date”, ”Win
Passed”,”Win In Progress”,”Win Rejected”,”Win Failed”, ”Linux
Passed”,”Linux In Progress”,”Linux Rejected”,”Linux Failed”,
”Network Passed”,”Network In Progress”,”Network
Rejected”,”Network Failed”], Instance=”diary_logbook_dev”)∧
(developer, create, Logtable[[#Data],[Status]]=[”Passed”, ”In
progress”,”Rejected”,”Failed”],Instance=”diary_logbook_dev” )∧
(developer, create, Logtable[[#Data],[Group]]=[”Windows”,
”Linux”, ”Network”], Instance=”diary_logbook_dev” )∧
(developer, read, Worksheet.name=”Logbook”,
Instance=”diary_logbook_prod” )∧
(developer, read, Worksheet.name=”Dashboard”,
Instance=”diary_logbook_prod” )∧
```

With the above description of the maintenance process and spreadsheet structural requirements, IT Administrator for Microsoft Windows servers access rules for ABAC4S protocol should be structured as follows:

```
(winadmin, update, Logtable[[#Data][Date]],
AND(Instance=”diary_logbook_prod”, Day=”Friday”)∧
(winadmin, update, AND(Logtable[[#Data],[IP
Address]]>”192.168.1.10”, Logtable[[#Data],[IP
Address]]<”192.168.1.100”), AND(Instance=”diary_logbook_prod”,
Day=”Friday”)∧
(winadmin, update, Logtable[[#Data],[Status]]=[”Passed”, ”In
progress”, ”Rejectd”, ”Failed”],
AND(Instance=”diary_logbook_prod”, Day=”Friday”)∧
(winadmin, update, Logtable[[#Data],[Group]]=”Windows”,
AND(Instance=”diary_logbook_prod”, Day=”Friday”)∧
(winadmin, read, Worksheet.name=”Logbook”,
Instance=”diary_logbook_prod”)∧
(winadmin, read, Worksheet.name=”Dashboard”,
Instance=”diary_logbook_prod”)
```

Similarly, IT Administrator for Linux servers access rules for ABAC4S protocol should be structured as follows:

```
(linuxadmin, update, Logtable[[#Data][Date]],
AND(Instance=”diary_logbook_prod”, Day=”Friday”)∧
(linuxadmin, update, AND(Logtable[[#Data],[IP
Address]]>”192.168.1.101”,Logtable[[#Data],[IP
Address]]<”192.168.1.200”), AND(Instance=”diary_logbook_prod”,
Day=”Friday”))∧
(linuxadmin, update, Logtable[[#Data],[Status]]=[”Passed”, ”In
progress”, ”In progress”, ”Failed”],
AND(Instance=”diary_logbook_prod”, Day=”Friday”))∧
(linuxadmin, update, Logtable[[#Data],[Group]]=”Linux”,
AND(Instance=”diary_logbook_prod”, Day=”Friday”))∧
(linuxadmin, read, Worksheet.name=”Logbook”,
Instance=”diary_logbook_prod”)∧
(linuxadmin, read, Worksheet.name=”Dashboard”,
Instance=”diary_logbook_prod”)
```

Finally, IT Administrator for networking infrastructure access rules for ABAC4S protocol should be structured as follows:

```
(netadmin, update, Logtable[[#Data][Date]],
AND(Instance=”diary_logbook_prod”, Day=”Friday”)∧
(netadmin, update, AND(Logtable[[#Data],[IP
Address]]>”192.168.1.201”,Logtable[#Data],[IP
Address]]<”192.168.1.300”), AND(Instance=”diary_logbook_prod”,
Day=”Friday”))∧
(netadmin, update, Logtable[[#Data],[Status]]=[”Passed”, ”In
progress”, ”In progress”, ”Failed”],
AND(Instance=”diary_logbook_prod”, Day=”Friday”))∧
(netadmin, update, Logtable[[#Data],[Group]]=”Network”,
AND(Instance=”diary_logbook_prod”, Day=”Friday”))∧
```

```
(netadmin, read, Worksheet.name="Logbook",
Instance="diary_logbook_prod")∧
(netadmin, read, Worksheet.name="Dashboard",
Instance="diary_logbook_prod")
```

With the above set of ABAC4S access rules for four different user groups, user's behaviors within diary logbook spreadsheet program are fully controlled and congruent with defined business process. In addition, all specified spreadsheet structural requirements are satisfied with the specified ABAC4S access rules. With combination of the environment attributes, complex user's roles and various timing constraints can be modelled for development and production instances. The worksheet "Logbook" in developed spreadsheet "diary_logbook_prod.xlsx" is presented in Figure 18.

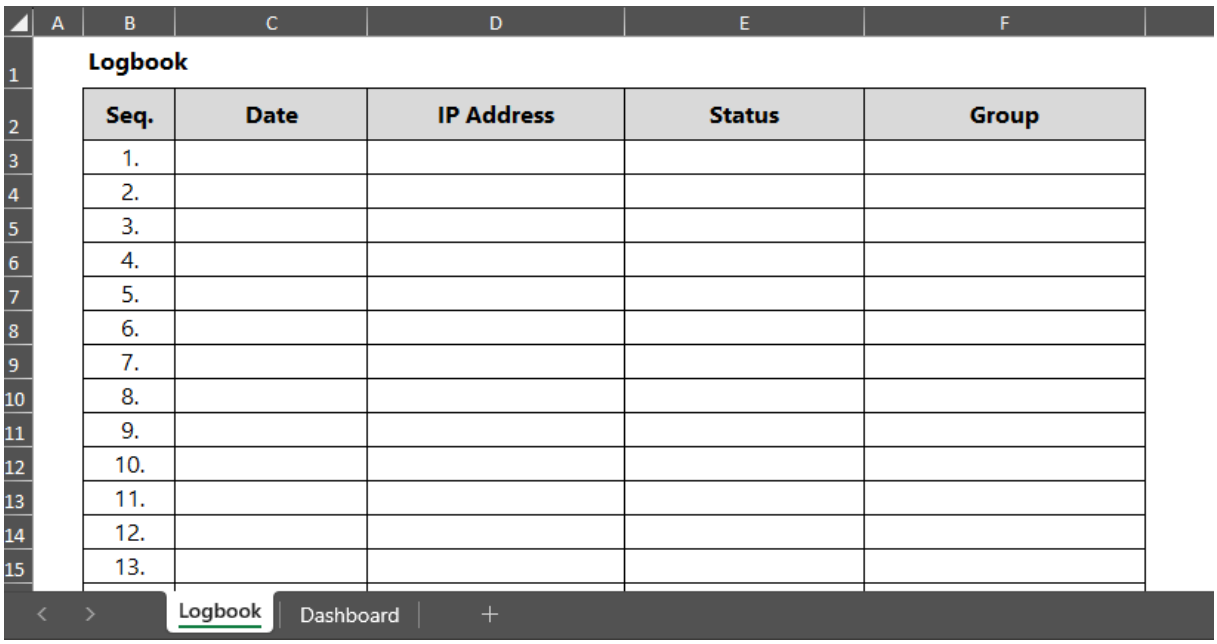

Figure 18. Logbook Worksheet

The worksheet "Dashboard" in developed spreadsheet "diary_logbook_prod.xlsx" is presented in Figure 19.

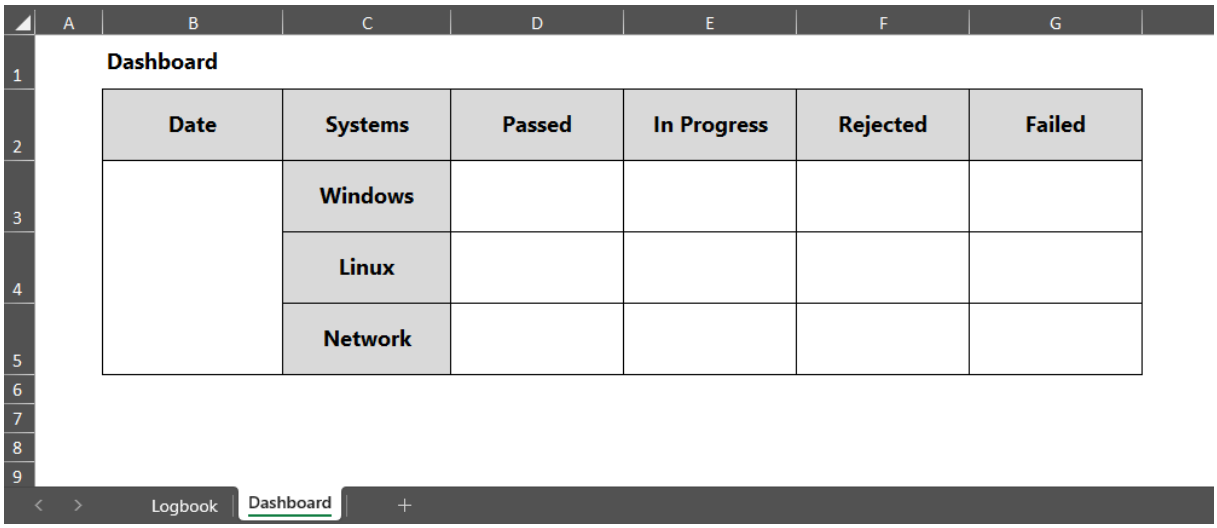

Figure 19. Dashboard Worksheet

8.2. Calibrations of Sensors in Analytical Laboratory

Negative Temperature Coefficient (NTC) probe calibration spreadsheet is used daily within analytical laboratory. Every Monday, a calibration expert performs checks of all 65 NTC probes utilized in the laboratory. Dedicated excel spreadsheet file is used for each NTC probe due to specific regulatory requirement. instance is managed for each calibration probe. In case, NTC probe does not perform satisfactory, calibration expert initiates calibration procedure. For cheaper NTC probes with lower accuracy, three-point Steinhart–Hart equation is used to perform NTC probe calibration [62]. For expensive and accurate NTC probes, more complex calibration procedure is performed with multiple measuring points used for Steinhart–Hart equation. After completion of all NTC probes checks and calibration procedures, Laboratory Manager reviews all calibration spreadsheets and if results of NTC probe calibration comply with laboratory guidelines, Manager changes color of result cell and corresponding worksheet to green as evidence of review and compliant status of NTC probes. Each Wednesday, the laboratory administrator edits header labels and adds information about probe serial numbers and information about calibration equipment used for NTC calibration. After completion, the laboratory administrator prints calibrations results on preformatted stickers and attaches them on probe housing. Based on the provided process description in place for calibration of NTC probes, following spreadsheet structural requirements and user's roles are defined:

- Spreadsheet developer should perform coding and development activities on separate development instance named "ntc_calibration_dev" and after successful testing, this

development instance should be merged to production instance named “ntc_calibration_prod”.

- NTC calibration spreadsheet should contain two worksheets. First worksheet named “NTC”, should contain input fields to enter NTC probe serial number and details of calibration equipment used. Worksheet “NTC” should also contain three input fields to enter measured temperatures and three input fields to enter measured NTC Probe resistances. The last part of the worksheet NTC should be three fields presenting calculated resulting Steinhart-Hart coefficients. All input fields should have associated labels in the left column of input value and physical unit labels in right column of input values. All input cells should be highlighted in light yellow color. All output cells should be highlighted in light grey representing status before managerial review and in light green color if laboratory manager confirms valid status of affected NTC probe. tIn second worksheet named “Calculation”, all calculation steps and intermediate formulas should be entered reference with named ranges. Absolute and relative cell references should be avoided. Links between input cells at worksheet NTC and calculation cells should be also constructed with named ranges for all affected cells. All calculations should be performed at the worksheet Calculation with results returned to worksheet NTC with reference to named ranges representing determined Steinhart–Hart coefficients.
- Users with role of spreadsheet developer assigned are allowed to change and modify layout, structure and content of “ntc_calibration_dev” development instance of NTC calibration spreadsheet. Spreadsheet developers are allowed to create and modify worksheet Calculation on development instance of spreadsheet with all formulas used to determine resulting coefficients. Formulas should be modularized to reflect calculation steps required to determine final coefficients. After successful development and testing, developer should create production instance “ntc_calibration_prod” that will be further used for laboratory calibration processes. In production instance “ntc_calibration_prod”, developer should have view only authorizations.
- Users with role of manager assigned should have update authorization on production instance “ntc_calibration_prod” to modify background color of output cells in the NTC worksheet. Allowed values for background color of output cells are light gray and light green, with light green representing evidence of successful managerial review and approval of calibration results.

- Users with role of analyst should have update authorizations of input fields for temperatures and resistance on production instance “ntc_calibration_prod”. To ensure that the calibration process follows defined standards for temperature ranges, allowed temperature ranges for three calibration points are as follows:
    - Temperature T1 allowed range: 0 - 10 °C.
    - Temperature T2 allowed range: 20 - 30 °C.
    - Temperature T3 allowed range: 80 - 90 °C.
- Users with role of administrator assigned should have update authorizations of input fields for NTC probe serial number and calibration equipment on production instance “ntc_calibration_prod”. Administrative work on calibration spreadsheets is allowed on Wednesday to ensure that both laboratory analysts and managers have enough time to complete their tasks and all NTC probes are calibrated and ready for laboratory processes during Thursdays and Fridays.

With the above description of the NTC probe calibration process and spreadsheet structural requirements, developer access rules for ABAC4S protocol should be structured as follows:

```
(developer, create, Worksheet.name=”NTC”,
Instance=”ntc_calibration_dev”)∧
(developer, create, Worksheet.name=”Calculation”, Instance=”
ntc_calibration_dev”)∧
(developer, create, NTC.C2.name=”Serno”, Instance=”
ntc_calibration_dev”)∧
(developer, create, NTC.C3.name=”cal_equipment”, Instance=”
ntc_calibration_dev”)∧
(developer, create, NTC.C6.name=”T1”, Instance=”
ntc_calibration_dev”)∧
(developer, create, NTC.C7.name=”T2”, Instance=”
ntc_calibration_dev”)∧
(developer, create, NTC.C8.name=”T3”, Instance=”
ntc_calibration_dev”)∧
(developer, create, NTC.C11.name=”R1”, Instance=”
ntc_calibration_dev”)∧
```

(developer, create, NTC.C12.name=”R2”, Instance=”
ntc_calibration_dev”)∧
(developer, create, NTC.C13.name=”R3”, Instance=”
ntc_calibration_dev”)∧
(developer, create, NTC.C17.name=”CoeffA”, Instance=”
ntc_calibration_dev”)∧
(developer, create, NTC.C18.name=”CoeffB”, Instance=”
ntc_calibration_dev”)∧
(developer, create, NTC.C19.name=”CoeffC”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C2.name=”T0”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C3.name=”L1”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C4.name=”L2”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C5.name=”L3”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C6.name=”Y1”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C7.name=”Y2”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C8.name=”Y3”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C9.name=”Z2”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C10.name=”Z3”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C11.name=”C”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C12.name=”B”, Instance=”
ntc_calibration_dev”)∧
(developer, create, Calculation.C13.name=”A”, Instance=”
ntc_calibration_dev”)∧

```
(developer, read, Worksheet.name=”NTC”,
Instance=”ntc_calibration_prod” )∧
(developer, read, Worksheet.name=”Calculation”,
Instance=”ntc_calibration_prod”)
```

With the above description of the NTC probe calibration process and spreadsheet structural requirements, manager access rules for ABAC4S protocol should be structured as follows:

```
(manager, update, Worksheet.name=”NTC”,
Instance=”ntc_calibration_prod” )∧
(manager, update, NTC.C17.backgroud_color=[”LightGrey”,
”LightGreen”], Instance=”ntc_calibration_prod”)∧
(manager, update, NTC.C18.backgroud_color=[”LightGrey”,
”LightGreen”], Instance=”ntc_calibration_prod”)∧
(manager, update, NTC.C19.backgroud_color=[”LightGrey”,
”LightGreen”], Instance=”ntc_calibration_prod”)∧
(manager, read, Worksheet.name=”Calculation”,
Instance=”ntc_calibration_prod”)
```

With the above description of the NTC probe calibration process and spreadsheet structural requirements, analyst access rules for ABAC4S protocol should be structured as follows:

```
(analyst, update, Worksheet.name=”NTC”,
Instance=”ntc_calibration_prod”)∧
(analyst, update, TYPE(T1)=”Number”, Instance=”
ntc_calibration_prod”)∧
(analyst, update, TYPE(T2)=”Number”, Instance=”
ntc_calibration_prod”)∧
(analyst, update, TYPE(T3)=”Number”, Instance=”
ntc_calibration_prod”)∧
(analyst, update, TYPE(R1)=”Number”, Instance=”
ntc_calibration_prod”)∧
(analyst, update, TYPE(R2)=”Number”, Instance=”
ntc_calibration_prod”)∧
```

```
(analyst, update, TYPE(R3)=”Number”, Instance=”
ntc_calibration_prod”)∧
(analyst, read, Worksheet.name=”Calculation”,
Instance=”ntc_calibration_prod”)
```

Finally, administrator access rules for ABAC4S protocol should be structured as follows:

```
(administrator, update, Worksheet.name=”NTC”,
Instance=”ntc_calibration_prod”)∧
(administrator, update, TYPE(Serno.value)=”String”,
AND(Instance=” ntc_calibration_prod”, Day=”Wednesday”)∧
(administrator, update, TYPE(cal_equipment.value)=”String”,
AND(Instance=” ntc_calibration_prod”, Day=”Wednesday”))∧
(administrator, print, Worksheet.name=”NTC”, AND(Instance=”
ntc_calibration_prod”, Day=”Wednesday”))
```

In case of administrator access rules, an example of extension to CRUD actions is demonstrated. Administrator has action “print” specified in access rules which permits printing of worksheet NTC on instance “ntc_calibration_prod” every Wednesday. Worksheet NTC in developed spreadsheet “ntc_calibration_prod.xlsx” is presented in Figure 20.

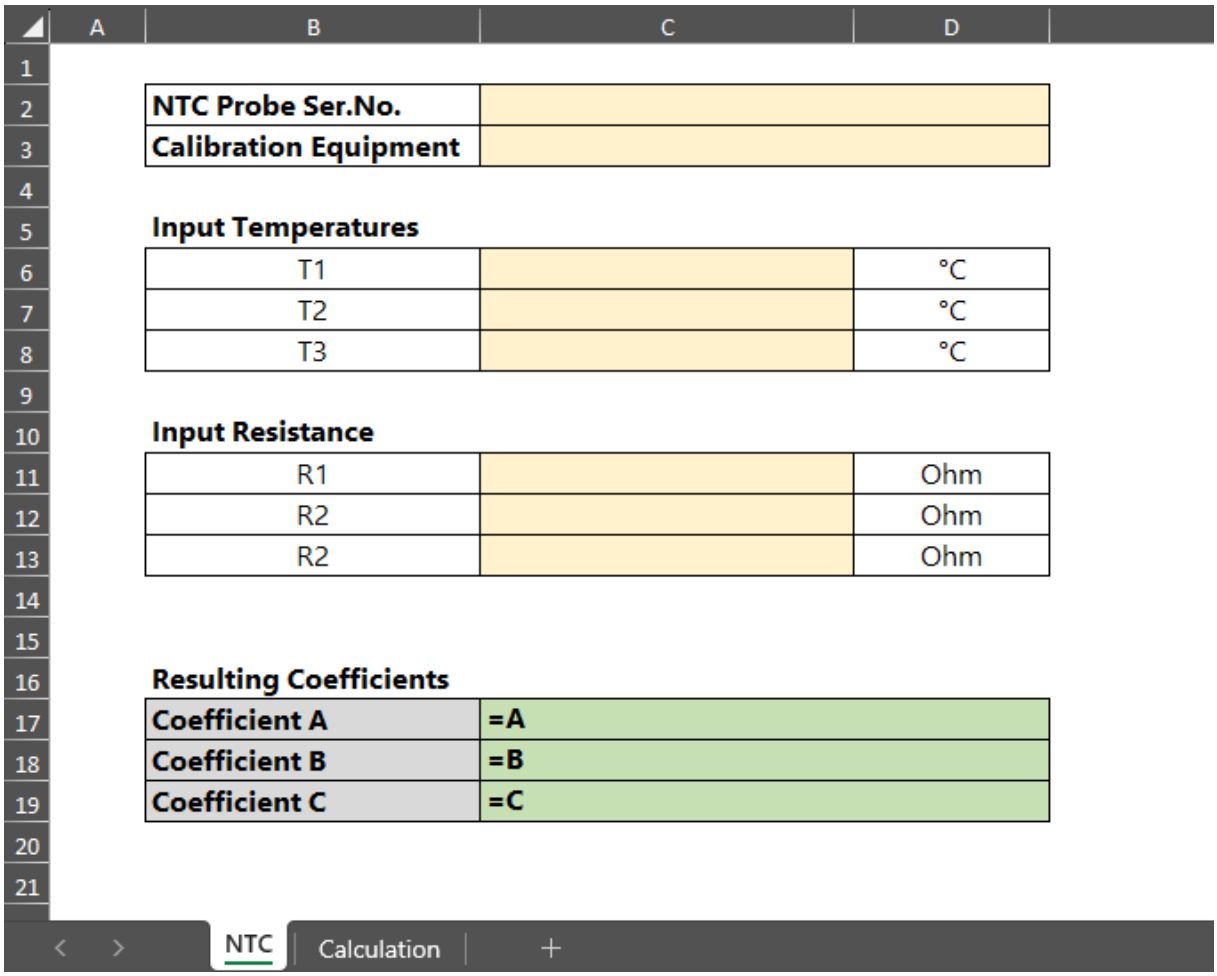

Figure 20. NTC Worksheet

Worksheet Calculation in developed spreadsheet "ntc_calibration_prod.xlsx" is presented in Figure 21.

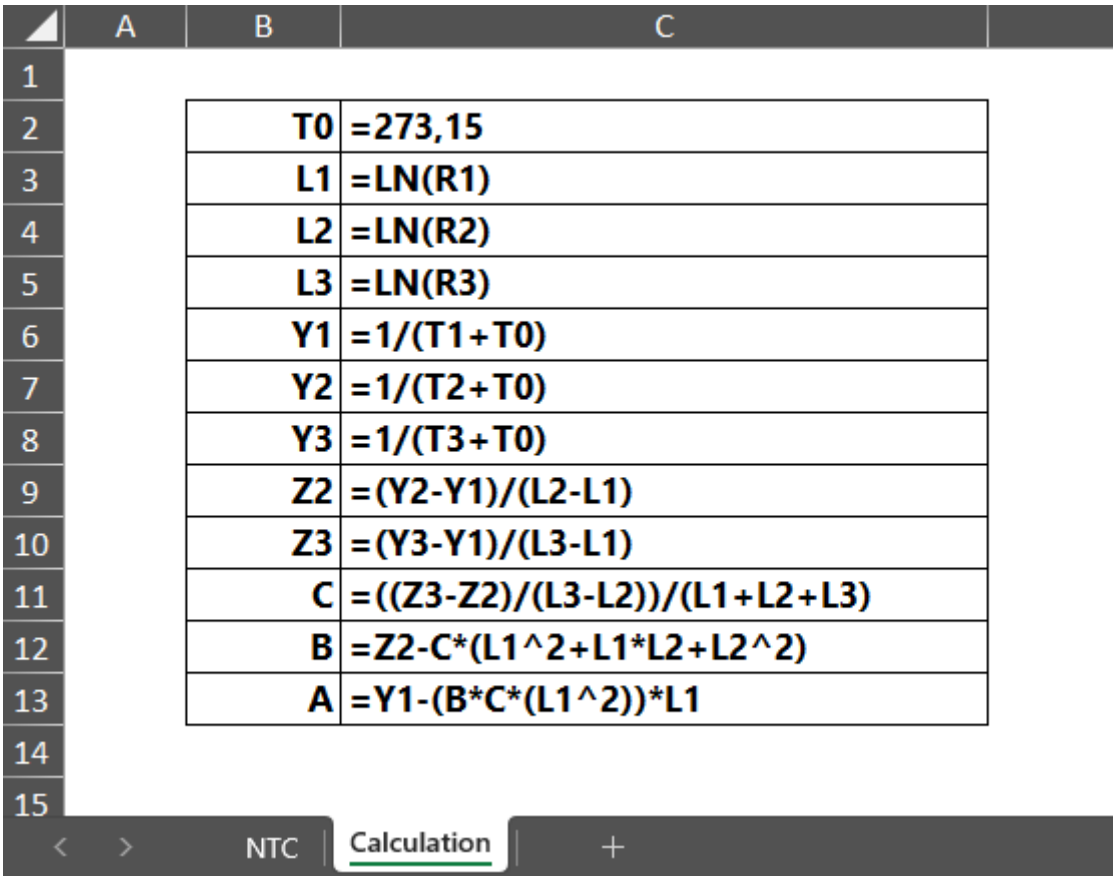

Figure 21. Calculation Worksheet

For the ABAC4S access rules definitions presented in both use cases, users' authorization and structural spreadsheet resource properties are denoted as abstract comma separated quadruples. These abstract data structures 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. In Appendix C., ABAC4S access rules defined in both cases are presented in JSON notation, commonly used as message exchange format in enterprise IT systems.

8.3. Users' Satisfaction with ABAC4S Protocol

Both cases of ABAC4S protocol were implemented during the period of 6 months. The author of the research presented in this thesis acted as facilitator and trainer for all participants with various spreadsheet roles during use cases with ABAC4S protocol practical implementation. In total 59 spreadsheet users participated during the period of 6 months. To evaluate spreadsheet users' satisfaction with practical ABAC4S implementation and document

valuable lessons learned at the end of the experiment, simple questionnaire with following three questions has been collected from all spreadsheet users:

- How satisfied were you with the use of ABAC4S protocol for your spreadsheets? (response on 1-5 scale, where 1 equal "very dissatisfied" and 5 equal "very satisfied").
- List positive examples, how ABAC4S protocol improved your work and overall experience with spreadsheets? (open-ended question according to spreadsheet user preferences).
- List negative examples, how ABAC4S protocol impacted negatively your work and overall experience with spreadsheets? (open-ended question according to spreadsheet user preferences).

All 59 spreadsheet users that participated in both uses cases with various roles specified with ABAC4S access rules provided answers to formulated questions. To the first question administered, majority of participant answered the question either a 4 or 5 (23 answers for "satisfied" and 34 answers for "very satisfied"), indicting an overall positive experience with ABAC4S protocol for spreadsheets. Two spreadsheet users answered the first question with 1 ("very dissatisfied") indicating their disagreement with overall spreadsheet use for business processes presented in two use cases. Specifically, both spreadsheet users indicated that full featured Enterprise Resource Planning (ERP) system should be used as a replacement for spreadsheet use in organizations.

The second and third questions were open-ended questions that did not offer a predetermined set of answers, allowing spreadsheet users to answer in their own words positive and negative experience with use of ABAC4S protocol for their spreadsheets. All collected answers are listed in the following tables, without any specific order or rank associated with collected answers. The author of this research considers collected responses a valuable source of user's feedback for future research. Positive experience with ABAC4S and collected answers to the second question are presented in Table 7.

Table 7. Positive user experience with ABAC4S protocol

No.	Positive user experience with ABAC4S protocol
1.	First impressions about ABAC4S access rule's structure were negative, however after a short period of usage and few access rules created from provided templates, initial frustration vanished. In contrast, creation or modification of access rules helps with documentation for actual job description.
2.	ABAC4S access rules increased users' satisfaction with laboratory processes and clarified roles and responsibilities that users must demonstrate during the use of spreadsheets.
3.	Maintaining authorizations in other IT systems and applications is far more complex and demands highly experienced IT administrator to transfer business roles to actual authorizations with special syntax for each system. ABAC4S access rules are simple to understand and maintain.
4.	ABAC4S protocol for automated quality assurance of spreadsheet programs improved compliance and regulatory status of analytical laboratory. Access rules were presented to external auditors as evidence of proper access control and segregation of duties between different users.
5.	Documented and evaluated access rules improved laboratory documentation and standard operating procedures.
6.	ABAC4S access rules facilitate and speed up onboarding of new employees. Their roles were documented with access rules and new employes could play on "sandbox" spreadsheets before they start using production versions of spreadsheets.
7.	ABAC4S protocol is invisible, and there are no changes in spreadsheet user interface. Provided feedback from ABAC4S protocol is very granular, so the user knows what must be corrected to return the spreadsheet in valid state.

Negative user's experience with ABAC4S and collected answers to the third question are presented in Table 8. Negative user experience with ABAC4S protocol

Table 8. Negative user experience with ABAC4S protocol

No.	Negative user experience with ABAC4S protocol
1.	ABAC4S protocol access rules are two complex to understand and maintain.
2.	Detective ABAC4S protocol implementation is too slow. After the last change submitted, feedback to the user should be provided faster, so that necessary corrections can be made in spreadsheets.

Important to note in relation to second negative comment is that performance of python scripts used for ABAC4S protocol computational logic was sometimes very slow. These scripts are not ready for production use in enterprise environments and have been used primarily as a proof-of-concept tool for the research presented in this thesis.

9. Conclusion and Further Research

In focus of the research presented in this thesis is automated quality assurance for spreadsheets. Specifically, this thesis is structured around the novel ABAC4S (Attribute Based Access Control for Spreadsheets) protocol designed for automated quality assurance of spreadsheets in multi-user environments. In Chapter 1, the research methodology is presented structured around design science research, formulated research goals and research hypothesis. Comprehensive descriptions of research phases and expected outcomes are provided as part of the defined research methodology. In Chapter 2, introduction to the spreadsheets is provided with examples of publicly documented spreadsheet horror stories that constitute motivation for research presented in this thesis. Chapter 3 provides a summary of related work in the field of taxonomies for spreadsheet errors, automated detection of spreadsheet errors and controlled access for spreadsheet users in modern enterprises. Afterwards, in Chapter 4, the ABAC4S protocol is presented with descriptions of model components and protocol rules. In Chapter 5, a brief introduction to model checking concepts is presented to establish foundation for Chapter 6 that provides formal verification of the proposed ABAC4S protocol with a symbolic model checker. In Chapter 7, research related to spreadsheet quality assurance is introduced with mapping of ABAC4S protocol characteristics to existing spreadsheet quality model and automated approaches to spreadsheet quality assurance. In Chapter 8 of this thesis, two use cases of ABAC4S protocol implementation are presented. The first case presents modeling of ABAC4S access rules to support logging of IT administrator activities related to management of key IT assets. In the second case, ABAC4S protocol was deployed to support calibration processes for NTC measurement probes in analytical laboratory. For both cases, the complete specifications of ABAC4S access rules are provided according to defined processes and user's job descriptions. Finally, this chapter provides critical discussions with reflection on conducted research, key findings and contributions to the spreadsheet research knowledge base. This thesis is accompanied by three appendices providing supplemental information to the research presented. In Appendix A., structured tables references are provided to support ABAC4S protocol access rules encoding presented in Chapter 4.5. In Appendix B., complete SMV source code for ABAC4S protocol is provided as part of model checking verification presented in Chapter 6. In Appendix C., examples of ABAC4S access rules in JSON format are provided to support developed access rules presented in Chapter 8. of this thesis.

To address the first research goal, novel ABAC4S protocol for automated quality assurance based on spreadsheet representation as a collection of resources has been developed. Defined

protocol addresses the need identified to control user's interaction with spreadsheets on granular level of spreadsheet resources in multi-user environments. In addition, ABAC4S protocol for automated quality assurance of spreadsheet programs uniquely addresses both methods in focus of quality assurance research, finding (detection) and avoiding (prevention) of spreadsheet errors and quality issues. The formal specification of novel ABAC4S protocol for spreadsheet has been provided with multi-faceted approach and combination of visual modeling, specification of protocol building blocks with algebraic data structures and direct graph representation.

To address the second research goal, defined ABAC4S protocol specifications has been verified for correctness with the model checking approach. Detailed steps and research journey from original ABAC4S protocol idea to formal specification of the protocol in applicable model checking language have been presented during this research phase. During construction of model checking specifications with SMV language, abstraction and refinement of model characteristics have been utilized to reduce model complexity and prevent state space explosion during verification with model checker. Representation with algebraic and graph data structures during ABAC4S protocol design and development have been instrumental during conversion to model checker SMV language and simulating transitions between spreadsheet states. Modules in SMV language have been designed to represent all possible state transitions with associated user's actions and hierarchical tree like representation of spreadsheet resources. Consequently, with state transitions for all possible combinations of user's actions and hierarchical representation of spreadsheet resources model checker verified all possible realistic scenarios where users in multi-user environments might have assigned roles of various complexities.

To address formulated research hypothesis, correctness property of the ABAC4S protocol has been evaluated with CTL temporal logic specification $AG\ (p \rightarrow AF\ q)$ (18). This CTL temporal logic specification evaluates all possible spreadsheet state changes for given user's roles. Model checking tool explores all possible traces in search of counterexample where desired property formulated with research hypothesis is not satisfied. As demonstrated in Chapter 6., NuSMV model checker evaluates CTL temporal logic specification to true, thus formally verifying that property formulated in research hypothesis holds, under assumption of correct SMV model specification.

Multiple challenges with model abstraction and state space explosion have been addressed during simulation with the model checking tool. Specifically, the reduction in ABAC4S protocol complexity based on single root path definition introduced in Chapter 4.3.4 of this

thesis has been instrumental for successful verification with a model checking approach without loss of model generality and correctness. Even though verification with model checker has been reduced to bare minimum representation of single root path for each spreadsheet resource, in case of 4 allowed actions and 4 different user roles, 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 on the state space explosion, in case the number of modeled actions and user roles increases to five, possible explorable state would grow to 25^7. This small increase in model complexity resulted in a more than 22 times larger model state space that must be explored with model checking tool.

Spreadsheet related research is a rich knowledge base with great scientific contributions. Results of the research presented in this thesis related to automated quality assurance for spreadsheets used 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 utilized to improve spreadsheet quality [22]. Research presented in this thesis contributes to spreadsheet research knowledge base with following:

- Novel ABAC4S protocol for automated quality assurance of spreadsheets in multi-user environments uniquely addresses both methods in focus of automated quality assurance research, finding (detection) and avoiding (prevention) of spreadsheet errors and quality issues.
- A multi-faceted approach to formal specification of ABAC4S protocol allows clear communication of research deliverables to thesis readers and other researchers focused on exciting research related to spreadsheets.
- Application of the model checking technique in verification of spreadsheet related research problems brings new perspective in spreadsheet research. This thesis presented modeling guidelines and insights into how to convert ABAC4S protocol specifications to the language accepted by the model checking tool.
- The ABAC4S protocol has been designed with user-centric approach to minimize impact on existing spreadsheet use in multi-user environments. This approach permits organizations to retain investment in their spreadsheets.
- Two use cases presented as part of the research suggest concrete ways for deploying ABAC4S protocol in other multi-user environments.

Despite advancements in understanding and mitigating spreadsheet risks, the dynamic and often ad hoc nature of spreadsheet life cycle phases continue to present challenges. Opportunities to further automate the generation of machine-readable user's access rules and implementation of ABAC4S protocol for automated quality assurance of spreadsheets in various multi-user environments will be explored in future research. Other possible directions for future research will include the integration of ABAC4S protocol with artificial intelligence methods for spreadsheet code suggestion and syntax reconstruction. Ultimately, ensuring spreadsheet quality is not merely a technical exercise but a critical component of informed and robust organizational functioning. I believe that ABAC4S protocol for automated quality assurance of spreadsheets in multi-user environments is one step in the right direction towards organizational cultures that prioritize spreadsheet quality as a fundamental aspect of data governance and decision-making.

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.

[37] S. Aurigemma, R. R. Panko, “The detection of human spreadsheet errors by humans versus inspection (auditing) software”, In Proc. Of EuSpRIG, 2010.

[38] R. Butler, “Is this spreadsheet a tax evader?”, Proceedings of the 33rd Hawaii International Conference on System Sciences, pp. 1–6, 2000.

[39] N. Kashmar, M. Adda, M. Atieh, H. Ibrahim, “A review of access control metamodels”, Procedia Computer Science, 184, 445-452., 2021.

[40] D. D. Downs, J. R. Rub, K. C. Kung, C. S. Jordan, “Issues in discretionary access control”, In 1985 IEEE symposium on security and privacy, pp. 208-208, IEEE, 1985.

[41] Y. Dholakia, “Mandatory Access Control – Problems in it and propose a model which overcomes them”, International Research Journal of Engineering and Technology (IRJET), (4)4, pp.2031-2035, 2017.

[42] E. O. Boadu, G. K. Armah, K. “Role-based access control (RBAC) based in hospital management”, Int. J. Softw. Eng. Knowl. Eng, 3, 53-67., 2014.

[43] C. Gross, “Announcing LAMBDA Helper Functions: Lambdas as arguments and more. https://techcommunity.microsoft.com/t5/excel-blog/announcing-lambda-helper-functions-lambdas-as-arguments-and-more/ba-p/2576648”, (June 10, 2024).

[44] P. Bartholomew, P. “Excel as a Turing-complete Functional Programming Environment”, arXiv preprint arXiv:2309.00115, 2023.

[45] “https://ecma-international.org/publications-and-standards/standards/ecma-376/”, ECMA International, [Accessed: Jun. 03, 2025].

[46] C. Hatmaker, “Reducing Errors in Excel Models with Component-Based Software Engineering”, arXiv preprint arXiv:2309.00650, 2023.

[47] Announcing Python in Excel: Combining the power of Python and the flexibility of Excel. “https://techcommunity.microsoft.com/t5/excel-blog/announcing-python-in-excel-combining-the-power-of-python-and-the/ba-p/3893439”, [Accessed: Jun. 03, 2025].

[48] T. Reschenhofer, B. Waltl, K. Shumaiev, F. Matthes, “A conceptual model for measuring the complexity of spreadsheets”, arXiv preprint arXiv:1704.01147, 2017.

[49] “https://support.microsoft.com/en-us/office/error-type-function-10958677-7c8d-44f7-ae77-b9a9ee6eefaa”, Microsoft Excel Error Types, [Accessed: Jun. 03, 2025].

[50] “https://support.microsoft.com/en-us/office/using-structured-references-with-excel-tables-f5ed2452-2337-4f71-bed3-c8ae6d2b276e”, Microsoft Excel structure references with Excel tables, [Accessed: Jun. 03, 2025].

[51] Peixoto, Christophe Campos, "Quality Model for Spreadsheets: Design and Implementation. MS thesis“, Universidade do Minho (Portugal), 2011.

[52] E. Aivaloglou, D. Hoepelman, F. Hermans, "A grammar for spreadsheet formulas evaluated on two large datasets." 2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM). IEEE, 2015.

[53] IEEE Computer Society, “IEEE Standard Classification for Software Anomalies”, In: IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993), pp. 1–23 (cit. on p. 1), 2010.

[54] G. Van Rossum, F. L. Drake, “Python 3 Reference Manual”, Scotts Valley, CA: CreateSpace, 2009.

[55] A. Hagberg, P. Swart, D. Chult, “Exploring network structure, dynamics, and function using NetworkX”, (No. LA-UR-08-05495; LA-UR-08-5495) Los Alamos National Lab. (LANL), Los Alamos, NM (United States), 2008.

[56] M. Vento, "A long trip in the charming world of graphs for pattern recognition". Pattern Recognition, 48(2):291–301., 2015.

[57] Zeina Abu-Aisheh, et al. "An exact graph edit distance algorithm for solving pattern recognition problems." 4th International Conference on Pattern Recognition Applications and Methods 2015. 2015.

[58] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals.”, Doklady Akademii Nauk SSSR, 163(4), 845–848., 1965.

[59] K. Riesen, S. Fankhauser, H. Bunke, “Speeding up graph edit distance computation with a bipartite heuristic.”, In Mining and Learning with Graphs, MLG 2007, Proceedings., 2007.

[60] “https://www.iso.org/quality-management/quality-assurance”, Quality assurance: A critical ingredient for organizational success, [Accessed: Jun. 03, 2025].

[61] M. Fowler. “Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston, MA, USA, 1999.

[62] Steinhart, John S., and Stanley R. Hart. "Calibration curves for thermistors." Deep sea research and oceanographic abstracts. Vol. 15. No. 4. Elsevier, 1968.

[63] Dietmar Jannach, et al. "Avoiding, finding and fixing spreadsheet errors–A survey of automated approaches for spreadsheet QA." Journal of Systems and Software 94 (2014): 129-150., 2014.

APPENDIXES

Appendix A. Structured Tables References

Syntax and for structured tables references are defined in table Table 9. [50].

Table 9. Structured Tables References

Item Specifier	Refers to:
#All	The entire table, including column headers, data, and totals (if any).
#Data	Just the data rows.
#Headers	Just the header row.
#Totals	Just the total row. If none exists, then it returns null.
#This Row or *@* or *@[Column Name]*	Just the cells in the same row as the formula. These specifiers can't be combined with any other special item specifiers. Use them to force implicit intersection behavior for reference or to override implicit intersection behavior and refer to single values from a column. Excel automatically changes #This Row specifiers to the shorter @ specifier in tables that have more than one row of data. But if your table has only one row, Excel doesn't replace the #This Row specifier, which may cause unexpected calculation results when you add more rows. To avoid calculation problems, make sure you enter multiple rows in your table before you enter any structured reference formulas.

.

Appendix B. SMV Source Code

```
MODULE spreadsheet_t()
VAR
    attributes:{s_a1,s_a2};
    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
    attributes:{a_a1,a_a2};
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};

MODULE named_object_t()
VAR
    attributes:{no_a1,no_a2};
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};

MODULE worksheet_t()
VAR
    attributes:{ws_a1,ws_a2};
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};
    table:table_t();
    cell:cell_t();

MODULE table_t()
VAR
    attributes:{t_a1,t_a2};
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};
    cell:cell_t();

MODULE cell_t()
VAR
    attributes:{c_a1,c_a2};
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};
```

```
    formula:formula_t();

MODULE formula_t()
VAR
    attributes:{f_a1,f_a2};
    role:{developer,tester,analyst,manager};
    a:{create,read,update,delete};

MODULE main
VAR
    spreadsheet:spreadsheet_t();

ASSIGN

next(spreadsheet.add_in.a) :=
    case
    (spreadsheet.role=spreadsheet.add_in.role) & \
(spreadsheet.a=read) & (spreadsheet.add_in.a in \
{update,create,delete}): read;
    (spreadsheet.role=spreadsheet.add_in.role) & \
(spreadsheet.a=update) & (spreadsheet.add_in.a in \
{read,create,delete}): update;
    (spreadsheet.role=spreadsheet.add_in.role) & \
(spreadsheet.a=delete) & (spreadsheet.add_in.a in \
{read,create,update}): delete;
    (spreadsheet.role=spreadsheet.add_in.role) & \
(spreadsheet.a=create) & (spreadsheet.add_in.a in \
{read,update,delete}): create;
    TRUE : spreadsheet.add_in.a;
    esac;


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

```
    (spreadsheet.role=spreadsheet.named_object.role) & \
(spreadsheet.a=create) & (spreadsheet.named_object.a in \
{read,update,delete}): create;
    TRUE : spreadsheet.named_object.a;
    esac;


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;


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

```
& (spreadsheet.a=create) & (spreadsheet.worksheet.a=create) &
(spreadsheet.worksheet.table.a in {read,update,delete}):
create;
    TRUE : spreadsheet.worksheet.table.a;
    esac;

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


next(spreadsheet.worksheet.cell.formula.a) :=
    case
    -- hierarchy resolution
    (spreadsheet.role=spreadsheet.worksheet.role) &\
(spreadsheet.worksheet.role=spreadsheet.worksheet.cell.role)&\
(spreadsheet.worksheet.cell.role=spreadsheet.worksheet.cell. \
formula.role) & (spreadsheet.a=read) & \
(spreadsheet.worksheet.a=read) & \
(spreadsheet.worksheet.cell.a=read) & \
(spreadsheet.worksheet.cell.formula.a in
{update,create,delete}): read;
```

```
    (spreadsheet.role=spreadsheet.worksheet.role) &\
(spreadsheet.worksheet.role=spreadsheet.worksheet.cell.role)&\
(spreadsheet.worksheet.cell.role=spreadsheet.worksheet.cell. \
formula.role) & (spreadsheet.a=update) & \
(spreadsheet.worksheet.a=update) & \
(spreadsheet.worksheet.cell.a=update) & \
(spreadsheet.worksheet.cell.formula.a in
{read,create,delete}): update;
    (spreadsheet.role=spreadsheet.worksheet.role) & \
(spreadsheet.worksheet.role=spreadsheet.worksheet.cell.role)&\
(spreadsheet.worksheet.cell.role=spreadsheet.worksheet.cell. \
formula.role) & (spreadsheet.a=delete) & \
(spreadsheet.worksheet.a=delete) & \
(spreadsheet.worksheet.cell.a=delete) & \
(spreadsheet.worksheet.cell.formula.a in
{read,create,update}): delete;
    (spreadsheet.role=spreadsheet.worksheet.role) & \
(spreadsheet.worksheet.role=spreadsheet.worksheet.cell.role)&\
(spreadsheet.worksheet.cell.role=spreadsheet.worksheet.cell. \
formula.role) & (spreadsheet.a=create) & \
(spreadsheet.worksheet.a=create) & \
(spreadsheet.worksheet.cell.a=create) & \
(spreadsheet.worksheet.cell.formula.a in
{read,update,delete}): create;
    TRUE : spreadsheet.worksheet.cell.formula.a;
    esac;
```

Appendix C. Example of ABAC4S Access Rules in JSON format

Examples of ABAC4S access rules defined in use cases are presented here in JSON format, commonly used as message exchange format in enterprise IT systems.

Table 10. Developer access rules in JSON format


```
[
    {
        "user": "developer",
        "action": "create",
        "Worksheet.name":"Logbook",
        "environment": {
            "instance": "diary_logbook_dev"
        }
    },
    {
        "user": "developer",
        "action": "create",
        "Worksheet.name":"Dashboard",
        "environment": {
            "instance": "diary_logbook_dev"
        }
    },
    {
        "user": "developer",
        "action": "create",
        "Logbook.Table.name":"Logtable",
        "environment": {
            "instance": "diary_logbook_dev"
        }
    },
    {
        "user": "developer",
        "action": "create",
        "Logbook.Table.name":"MaintenanceStatus",
        "environment": {
            "instance": "diary_logbook_dev"
        }
    },
    {
        "user": "developer",
        "action": "create",
```

```
        "Logtable[#Headers]": [
            "Seq",
            "Date",
            "IP Address",
            "Status",
            "Group"
        ],
        "environment": {
            "instance": "diary_logbook_dev"
        }
    },
    {
        "user": "developer",
        "action": "create",
        "MaintenanceStatus[#Headers]": [
            "Date",
            "Win Passed",
            "Win In Progress",
            "Win Rejected",
            "Win Failed",
            "Linux Passed",
            "Linux In Progress",
            "Linux Rejected",
            "Linux Failed",
            "Network Passed",
            "Network In Progress",
            "Network Rejected",
            "Network Failed"
        ],
        "environment": {
            "Instance": "diary_logbook_dev"
        }
    },
    {
        "user": "developer",
        "action": "create",
        "Logtable[[#Data],[Status]]": [
            "Passed",
            "In Progress",
            "Rejected",
            "Failed"
        ],
        "environment": {
            "Instance": "diary_logbook_dev"
```

```
        }
    },
    {
        "user": "developer",
        "action": "create",
        "Logtable[[#Data],[Group]]": [
            "Windows",
            "Linux",
            "Network"
        ],
        "environment": {
            "Instance": "diary_logbook_dev"
        }
    },
    {
        "user": "developer",
        "action": "read",
        "Worksheet.name":"Logbook",
        "environment": {
            "instance": "diary_logbook_prod"
        }
    },
    {
        "user": "developer",
        "action": "read",
        "Worksheet.name":"Dashboard",
        "environment": {
            "instance": "diary_logbook_prod"
        }
    }
]
```

Table 11. Manager access rules in JSON format

```
[
    {
        "user": "manager",
        "action": "update",
        "Worksheet.name":"NTC",
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    },
    {
```

```
        "user": "manager",
        "action": "update",
        "NTC.C17.backgroud_color":[
            "LightGrey",
            "LightGreen"
        ],
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    },
    {
        "user": "manager",
        "action": "update",
        "NTC.C18.backgroud_color":[
            "LightGrey",
            "LightGreen"
        ],
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    },
    {
        "user": "manager",
        "action": "update",
        "NTC.C19.backgroud_color":[
            "LightGrey",
            "LightGreen"
        ],
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    },
    {
        "user": "manager",
        "action": "read",
        "Worksheet.name":"Calculation",
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    }
]
```

Table 12. Analyst access rules in JSON format

```
[
    {
        "user": "analyst",
        "action": "update",
        "Worksheet.name":"NTC",
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    },
    {
        "user": "analyst",
        "action": "update",
        "TYPE(T1)":"Number",
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    },
    {
        "user": "analyst",
        "action": "update",
        "TYPE(T2)":"Number",
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    },
    {
        "user": "analyst",
        "action": "update",
        "TYPE(T3)":"Number",
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    },
    {
        "user": "analyst",
        "action": "update",
        "TYPE(R1)":"Number",
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    },
    {
        "user": "analyst",
```

```
        "action": "update",
        "TYPE(R2)":"Number",
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    },
    {
        "user": "analyst",
        "action": "update",
        "TYPE(R3)":"Number",
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    },
    {
        "user": "analyst",
        "action": "read",
        "Worksheet.name":"Calculation",
        "environment": {
            "instance": "ntc_calibration_prod"
        }
    }
]
```

Table 13. Administrator access rules in JSON format

```
[
    {
        "user": "administrator",
        "action": "update",
        "Worksheet.name":"NTC",
        "environment": {
            "operator": "AND",
            "criteria": [
                {
                "field": "instance",
                "operator": "=",
                "value": "ntc_calibration_prod"
                },
                {
                "field": "day",
                "operator": "=",
                "value": "Wednesday"
                }
            ]
```

```
        }
    },
    {
        "user": "administrator",
        "action": "update",
        "TYPE(Serno.value)":"String",
        "environment": {
            "operator": "AND",
            "criteria": [
                {
                "field": "instance",
                "operator": "=",
                "value": "ntc_calibration_prod"
                },
                {
                "field": "day",
                "operator": "=",
                "value": "Wednesday"
                }
            ]
        }
    },
    {
        "user": "administrator",
        "action": "update",
        "environment": {
            "operator": "AND",
            "criteria": [
                {
                "field": "instance",
                "operator": "=",
                "value": "ntc_calibration_prod"
                },
                {
                "field": "day",
                "operator": "=",
                "value": "Wednesday"
                }
            ]
        }
    },
    {
        "user": "administrator",
        "action": "print",
```

```
        "Worksheet.name":"NTC",
        "environment": {
            "operator": "AND",
            "criteria": [
                {
                "field": "instance",
                "operator": "=",
                "value": "ntc_calibration_prod"
                },
                {
                "field": "day",
                "operator": "=",
                "value": "Wednesday"
                }
            ]
        }
    }
]
```

CV

Miro Zdilar was born on 24th of June 1970 in Pula, Croatia. He finished elementary school "Vladimir Nazor" in Križevci, and technical school "Ruđer Bošković" in Zagreb, Croatia. He graduated in 1995 from the Faculty of Electrical Engineering and Computing of the University of Zagreb with the thesis cro. *"Analiza slike neuronskom mrežom"* (eng. *"Perceptron Neural Network for Pattern Recognition"*) and mentor Academic Professor Sven Lončarić, PhD. Miro Zdilar continued his research and postgraduate studying at the Faculty of Electrical Engineering and Computing of the University of Zagreb where he earned Master of Science degree in 2011 with the thesis cro. *"Transakcijski protokoli za uslužno usmjerenu arhitekturu"* (eng. *"Transaction protocols for service oriented architecture"*) and mentor Academic Professor Ignac Lovrek, PhD. During 2007 he enrolled in the in-company Master of Business Administration (MBA) study program at Rotterdam School of Management Erasmus University. He graduated in 2009 with the thesis *"Intranet Knowledge Sharing-Implementation of the GMP Training Reinforcement Pilot"* and mentor Professor H.P. Borgman, PhD. During 2019 he enrolled in the postgraduate doctoral study of Information Science, at the Faculty of Organization and Informatics of the University of Zagreb. He currently works as a Chief Information Security Officer (CISO) at the Raiffeisenbank Austria dd, Zagreb. His fields of interest are spreadsheet engineering, automated verification techniques and data analysis. He is married and has two children.

List of scientific publications

[1] Miro Zdilar, "Model Checking Access Control Protocol for Spreadsheets", Journal of information and organizational sciences, Vol. 49 No. 1, (2025), 39-52. https://doi.org/10.31341/jios.49.1.3

[2] Miro Zdilar, "Attribute Based Access Control Metamodel for Spreadsheet Programs." 35th International Scientific Conference CECIIS 2024. Varaždin: University of Zagreb, Faculty of Organization and Informatics, 2024.

[3] Miro Zdilar, "Towards Automated Detection of Qualitative Spreadsheet Errors in Multi-user Environments." 34th Central European Conference on Information and Intelligent Systems (CECIIS 2023), 2023.

[4] Miro Zdilar, “Transakcijski protokoli za uslužno usmjerenu arhitekturu.”, Dissertation University of Zagreb. Faculty of Electrical Engineering and Computing. Department of Telecommunications, 2011.

List of professional publications

[1] Miro Zdilar, “Revizija procesa razvoja sustava (SDLC)”, I. Konferencija internih revizora “Razvoj i izazovi interne revizije”, Opatija, 2009.