Theory of Hybrid Systems Informatik 2
RWTH AACHEN UNIVERSITY

The present work was submitted to the LuFG Theory of Hybrid Systems

BACHELOR OF SCIENCE THESIS

BUILDING AN E-LEARNING WEB APP WITH AUTOMATED EXERCISE GENERATION

Rui Miguel Oliveira Sobrinho

Communicated by
Prof. Dr. Erika Ábrahám

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr.-Ing. Ulrik Schroeder

Additional Advisor:
József Kovács

Aachen, 6.11.2023

Abstract

This paper presents the development of an e-learning web application, using React, with a focus on automated exercise generation. Our tool encourages users to learn concepts and enhance analytical thinking related to satisfiability checking.

To amplify user engagement and motivation, the website incorporates gamification elements including a leaderboard to foster competition. For a personalised learning experience, there will be an opportunity to try previously wrongly answered questions again. Selected exercises offer different levels of hints. The users may choose to solve the exercises with a time limitation and whether they want to do singular tasks or a set of tasks. This creates exam-like conditions and encourages users to think critically and efficiently solve the exercises like they would need to in an exam. The implementation of this e-learning web application, shall provide a platform for users to acquire knowledge and develop skills in satisfiability checking.

Contents

Chapter 1

Introduction

In the field of theoretical computer science, the satisfiability problem aims to solve the issue of whether a given formula in propositional logic has a model that satisfies a given formula [ÁNP23]. Explicitly this means, that the problem attempts to find out if the propositions of the formula can be substituted with certain truth values so that the whole formula evaluates to true. While this problem is decidable, it falls within the NP-complete class. In other words, it belongs to a set of problems, whose solutions can be verified in polynomial time, by a deterministic Turing machine.

Throughout the years, this branch of computer science gained a lot of popularity. It has seen the development of various methods and the creation of new algorithms to address the satisfiability problem. Moreover, the scope of this problem has expanded beyond propositional logic; it now encompasses formulas in first-order logic. Furthermore, problems from other branches of computer science can be translated logically to a satisfiability problem and thus be solved using the methods that are available. One such branch is model checking, where a finite state model is checked for the correctness of its behaviour [VWM15]. This is of relevance for both hardware and software systems. The main benefit of model checking over other verification approaches like simulation based verification is its exhaustiveness, as it checks every state and transition of the system. This gives it, and therefore also satisfiability checking, great demand in multiple industries. Other relevant fields for satisfiability checking are planning [NPM$^+$14] and scheduling [BCSV14].

The chair of Hybrid Systems of the RWTH which offers the course on Satisfiability Checking, takes a lot of factors into consideration while offering this subject. Thus they try to make it interactive and request active participation. In order to support the students, annotated slides, lecture recordings and bonus tests are also offered. So the students have a myriad of materials available to help them study on top of the lectures they can attend every week during the semester.

1.1 Motivation

Those bonus tests are the focus point of this thesis. We try to build a web application which provides the aforementioned tests. It is intended that the application presents

the tests in a user-friendly manner, enabling users to solve them effectively. After completion, it displays whether the given answer is correct, partially correct, or completely incorrect. Additionally specific tasks get the ability to show different kinds of hints to help the users. Features are added in order to motivate continued learning and practice. One such feature is an optional timer, which the user can activate to restrict the time available to do the exercise. This simulates exam-like conditions and the user can practise to solve the exercises under time pressure.

The goal of this application is to help the students by facilitating their learning possibilities and engage more into the subject of satisfiability checking. It shall motivate students to learn as well as enable and simplify learning possibilities for satisfiability checking. The large number of exercises available, aligned with the course supplied at RWTH, yields the possibility of learning a variety of different algorithms connected to the subject. The tasks that had hints added give an extra possibility to try to solve them in case a user gets stuck on that specific exercise. An added functionality where users are able to try previously wrongly answered questions helps them to further deepen their knowledge. As the display of hints is generalised, other tests can have hints added later in order to expand the offer of tasks with additional help.

1.2 Related Work

Most papers found about automated exercise generation for satisfiability checking were written here at RWTH for example [Fil, EE22], which cover exercise generation for specific topics like interval constraint propagation or real root isolation. A whole website with automated exercise generation with context to satisfiability checking was implemented by Prof. Dr. rer. nat. Johannes Waldmann at the HTWK Leipzig [Wal14]. Furthermore Prof. Kovács from the Vienna University of Technology (TU Wien) worked on automatically generating exercise sheets for online exams for their course "automated deduction", which also covers aspects of SAT/SMT solving [HKR21]. While both works generate exercises relating to satisfiability checking, [HKR21] does not handle automatic evaluation. On the other hand the exercises from [Wal14] is able to grade solutions semantically.

Chapter 2

Preliminaries

2.1 DPLL and CDCL

DPLL and CDCL are 2 algorithms used in satisfiability checking. DPLL stands for Davies-Putnam-Logemann-Loveland. The algorithm uses propositional logic formulas that have to be in conjunctive normal form (CNF) for the DPLL algorithm to be used.[Sin07]

A propositional logic formula is a syntactically well-formed formula which does not contain *existential* ($\exists$) or *universal* ($\forall$) quantifiers. It is composed of a finite number of *clauses*. Those clauses themselves are a finite set of atomic propositions which are *boolean variables* (a, b, c, ...). A single such variable is called a *literal*. A literal can also be in a negated form ($\neg$a). The negated form represents the complement, or the opposite, of its non-negated form. These literals can be used with logical operators, such as *and* ($\wedge$), *or* ($\vee$), *not* ($\neg$) and *implies* ($\rightarrow$). The propositions can be substituted with truth values so that the whole formula evaluates to either true (1) or false (0). The act of giving every variable a truth value is called an assignment.

A formula that can be evaluated to true is also called *satisfiable*, while one that cannot be, one that always evaluates to false no matter the assignment, is called *unsatisfiable*. A *tautology* is a formula that no matter the assignment, always evaluates to true.

A formula in CNF is a conjunction of clauses ($C_1 \wedge C_2 \wedge ... \wedge C_m$) which themselves are a disjunction of literals ($a_1 \vee a_2 \vee ... \vee a_n$). So only $\wedge$, $\vee$ and $\neg$ are present in such a formula and have the following shape, with Y_{ij} being literals:

$$\bigwedge_{n=1}^{n} \bigvee_{j=1}^{m} Y_{ij}.$$

A clause can be in one of 4 states:

- it can be satisfied, if at least one literal in the clause is true,
- it can be unsatisfied, if all literals in the clause are false,
- it can be unit, if all literals but one are false,
- it is unresolved in every other case.

The DPLL-algorithm uses Boolean Constraint Propagation (BCP)[Tin02] [ÁNP23] to check if the formula contains a unit clause. If such a unit clause is found a truth value is propagated, as there is only one value that fits in order to satisfy the formula. The truth value of that literal in question is stored and every clause containing the literal of the unit clause is deleted as it automatically is satisfiable. This way, the algorithm simplifies the starting formula so that it determines faster if the whole formula is satisfiable or not. Every clause containing the negated literal (so simply a literal that is assigned false in this case) also removes the corresponding literal in order to simplify the formula. If no more BCP can be done the algorithm looks if it contains an empty clause. This occurs if by only applying BCP a clause gets all its literals removed, so all its literals assigned to false. Then the algorithm returns unsatisfiable. If on the contrary, there are no more unresolved clauses left, a satisfying assignment has been found and the algorithm returns that the formula can be satisfied.

If neither is the case the algorithm hasn't concluded yet, there are clauses that are unresolved so it chooses a literal and assigns a truth value to it. This isn't a final decision yet, as the algorithm may backtrack and try the other truth value if it doesn't find a satisfying assignment for the whole formula. After deciding on a value for the chosen literal it checks whether it can apply BCP. The application of BCP and choosing itself a truth value (instead of being forced to attribute one) is continued until the algorithm comes to a conclusion of the taken path. Such a conclusion is either finding a satisfiable assignment and returning satisfiability or detecting a conflict. In the latter case it backtracks to the last time a decision was made, and chooses the other option and explores the path from there. If at some point a satisfiable assignment is found, it ends and returns that the formula can be satisfied. Otherwise, if every possible path results in unsatisfiability, that becomes the end result as no possible assignment can satisfy the formula.

A conflict happens, when a clause implies that a certain literal has to be set to a certain value but that same literal already has a different assignment; either by BCP or decision. In regard to such a conflict, the clause can't be satisfied and thus the whole formula can't be satisfied. If possible, it backtracks to the last time a value was given by decision (and not implication) and it chooses the other truth value and starts anew from there. The algorithm finishes by either finding a satisfiable assignment, or finding out that no such assignment exists and therefore the formula is unsatisfiable. The above mentioned process is formalised in pseudo code:

```
1   bool DPLL(CNF_formula φ){
2       while(φ contains unit clause (literal l)){
3           assign true to l
4           delete clauses containing l
5           delete ¬l from all other clauses
6       }
7       if (φ is empty) return true
8       if (φ contains empty clause) return false
9       choose literal l in φ and assign a truth value
10      return DPLL(φ∪{l}) or DPLL(φ∪{¬l})
11  }
```

Example. Let's look at how the algorithm works in the following example:

$$(a \vee b) \wedge (\neg b) \wedge (c \vee d) \wedge (b \vee \neg c \vee d)$$

The second clause is a unit clause, so we are forced to assign $false$ to b in order to satisfy $\neg b$. This causes propagation to continue and assign $true$ to a, otherwise the first clause can't be satisfied. Following that, the 2 remaining clauses are practically the same, as b can be ignored in the fourth clause and thus we have $(c \vee d)$ twice. The algorithm would see it as such, as it doesn't consider b to be in the last clause anymore after deleting it. As no propagation is possible here, one literal is chosen, for example c and a truth value gets assigned to it. Now all clauses are satisfied and we know that the starting formula is satisfiable.
As we don't have a full assignment yet, because d didn't receive a truth value, it can be assigned any of the possible 2 values.

Nowadays almost anytime when DPLL is used, it is extended with CDCL which stands for Conflict-Driven Clause Learning [Oh16, ÁK16]. The main difference between CDCL and DPLL is that CDCL's back jumping does not have to be chonological. As an example, let's say we have variables $x_1, x_2, ..., x_n$ that are all assigned to true. The algorithm backtracks the chain of implications and applies resolution in order to get a reason for the conflict. There it obtains a so-called conflict clause which is added to the clause set. Resolution is used to find the unsatisfiable core of the unsatisfiable formula. Resolution is done in the following way: we have clauses $(a, a_1, ..., a_n)$ and $(\neg a, b_1, ..., b_m)$. Through Binary Resolution, both clauses are combined and the literals a and $\neg a$ cancel out. The new clause we get is the combined rest of both clauses, so $(a_1, ..., a_n, b_1, ..., b_m)$.

$$\frac{c_1 : (a, a_1, ..., a_n) \qquad c_2 : (\neg a, b_1, ..., b_m)}{c_l : (a_1, ..., a_n, b_1, ..., b_m)}$$

The new clause is called the resolvent of the clauses c_1 and c_2. It can be shown that resolution is a purely logical outcome: if a in c_1 is true, then at least one of the literals from b_1 to b_m has to be true in order to satisfy c_2. Coincidentally: if a is false, then $\neg a$ from c_2 is true and c_2 is satisfied, so at least one literal from a_1 to a_n has to be true to satisfy c_1. Conclusively, at least one of a_i or b_j has to be true in order to satisfy the clauses, which is shown by the common clause C_l.

Let it be in this case that the algorithm learned the clause $(\neg x_5, \neg x_{10}, \neg x_{15})$ after ending up in a conflict. This shows us that at least one of these variables should have been assigned false. If in this example, the value of these 3 variables was decided instead of propagated (assuming it was done in chronological order according to their numbers), this brings the possibility forward to not just go back to where x_{15} was decided but already to x_{10} or even x_5. At x_{10}, it can then be decided in a way that the clause becomes unit and therefore propagation can take place in order to solve the conflict and maybe find a satisfying assignment.

2.2 Virtual substitution and generating test candidates

The virtual substitution method constructs a finite set $T \subset \mathbb{R}$ of test candidates with $\exists x_1 ... \exists x_n \varphi \equiv \exists x_1 ... \exists x_{n-1} \bigvee_{t \in T} \varphi[t//x_n]$, for real algebraic (rational and non zero) formula $\exists x_1 ... \exists x_n \varphi$ with $n > 0$ and φ quantifier-free [CÁ11, Košˇ16]. Here $[A//B]$ stands for virtually substituting A for B. Additionally T contains representative points from sign-invariant regions. To compute those regions, we need to determine the real roots of univariate polynomials. This can be done by making use of solution equations up to a polynomial degree of 4.
A general polynomial has the form of $ax^2 + bx + c \in \mathbb{Z}[x]$ and the roots are as follows:

- If the polynomial is constant in x, so if $a = 0 \wedge b = 0 \wedge c = 0$, all real numbers are zero,
- If the polynomial is linear in x, so if $a = 0 \wedge b \neq 0, \xi_0 = -\frac{c}{b}$ is the real root,

If the polynomial is quadratic in x, there are 2 solutions:

- If $a \neq 0 \wedge b^2 - 4ac \geq 0$ then $\xi_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ is the first solution and real zero of the polynomial and
- if $a \neq 0 \wedge b^2 - 4ac > 0$, then $\xi_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$ is the second root of the quadratic polynomial.

Multivariate polynomials are polynomials with multiple variables. They are also able to be solved in the same way as they can be seen as univariate polynomials with polynomial coefficients. So we get the same solution equations but parameterised in value of variables in the coefficients.
As an example:
The polynomial $3z^2yx^2 - 5zy^3x + 2zy - 7 \in \mathbb{Z}[z,y,x]$
can be seen as: $\boldsymbol{3z^2y}x^2 - \boldsymbol{5zy^3}x + \boldsymbol{2zy - 7} \in \mathbb{Z}[\boldsymbol{z,y,a}][x]$ with the coefficients: $a = 3z^2y, b = 5zy^3, c = 2zy - 7$

We are looking for the roots of the polynomial, so where the polynomial is zero. In the same sense, we can compare the polynomials to zero and try to find out if it satisfies a given constraint $(=, <, >, \leq, \geq, \neq)$ and in case of unsatisfiability, give an explanation as to why. For a polynomial to change the sign of its solution, it has to "pass through" zero. So if we know the roots of the polynomial, we can detect the regions where the solutions of the polynomial have a specific sign.

As mentioned before, the polynomial can be constant or have no real zeroes. In that case, the sign does not change and we can use any possible real number to find out what the sign is for that (in)equation in the whole $\mathbb{R}$ (or rather domain, if the domain is not $\mathbb{R}$).

In cases where the polynomial is not constant, we have real zeros and we can find the solution intervals. Those zeros mark the endpoints of the intervals where the sign does not change. The zeroes are included in the solution interval in the cases where

the constraints are inclusive to zero ($\leq, \geq$) and excluded in the other case ($<, >$). In the case where we handle an equation instead of an inequation ($=$), the solutions are the roots of the polynomial and not an interval.

We can also have multiple polynomial constraints in which case we construct a possible common solution interval. In general, we can represent each candidate solution interval by its left-most point. If the interval is closed, that leftmost point is the endpoint of that interval whereas if the interval is open, then we take the leftmost point plus a very small (infinitesimal) value ϵ, so that the resulting element is in the solution interval.

Example. Let's look at the following example and how to solve it.

Besides $(-\infty)$, which of the following expressions are generated as test candidates to eliminate x from:
$$(y - 2 \geq 0 \wedge 4x + 4 = 0 \wedge 4x - 4y < 0)?$$

The first constraint doesn't contain x, so we do not have to look at that one. The second one does. As it is a natural common equation, it is simple to find out that the solution is -1. Thus we have our first possible test candidate. The third expression also contains x, so we'll have another possible test candidate. After ordinary arithmetic, we can simplify the inequation to $x < y$. As we have $<$ and not $\leq$, we cannot simply use y as a test candidate and have to add an infinitesimal value to it. So the second and last possible test candidate for this example is $y + \epsilon$.

Chapter 3

Implementation

3.1 Preprocessing

The architecture of the web project can be divided into three main components: a frontend, a backend and a database. The tests to be solved by the users are automatically generated by different python programs. Each of those programs is responsible for a specific test and generates an XML file, where a chosen number of instances (mostly 300 or 500) of that test are generated. As these exercise generators were already provided, a simple python script was written to convert the XML files into JSON files. This adaptation proved essential to streamline development, as JSON can be readily parsed by a standard JavaScript function, whereas XML necessitates the use of an additional parsing tool.

Another detail to facilitate the implementation is the naming of the files. It follows a certain structure, namely "moodleX.json", where X is a number. This allows to differentiate the tasks accordingly and save the tests the users do.

3.2 Frontend

For the frontend of this website React (also known as React.js) was used [reaa, reab]. React is a JavaScript library with the main objective of building single page applications by using individual pieces called components. It is maintained by Meta, formerly known as Facebook. In addition to that, Node.js is used with the npm package manager [nodb, noda, npmb, npma].

Upon initiating the web page users are prompted to either log in, or if not yet registered, navigate to the sign up page to create an account. The need for user accounts is necessary to facilitate the feature allowing users to reattempt previously failed tests. Furthermore, it allows to save the data necessary to have a leaderboard.

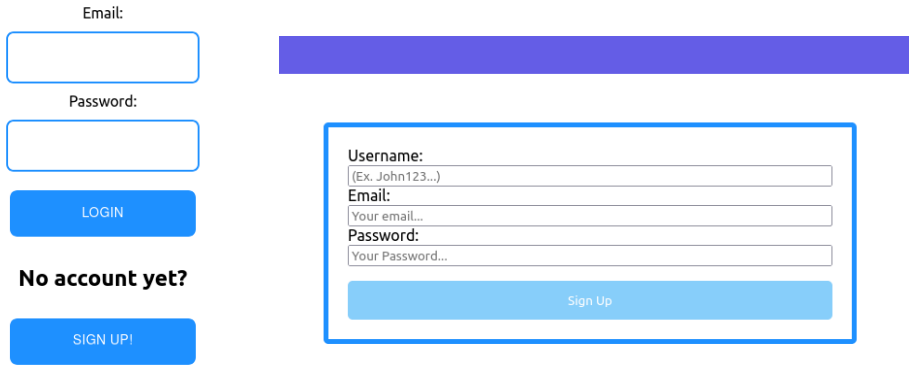
The sign up page makes use of formik and yup [for, yup]. Formik serves as a tool for constructing forms within the React framework, and here, it is utilised to create the sign-up form. On the other hand, Yup is a schema builder which is used for value parsing and validation during runtime. Consequently it ensures that the user inputs

a valid email address, username and password. The data is sent to the backend which then saves it to the database.

The login page requires users to provide an email address and password to then check with the database if the user exists and the correct password was used to log him in. Authorisation and authentication is handled in the backend. Figure 3.1 shows the login and sign up page of the website.

Within the home component, several dropdown buttons are available for the different tests. This is done to not have dozens of buttons that show each specific test individually. The tests can be grouped to any arbitrary criteria, for example according to the chapters in the lecture. In this instance, the tests have been grouped in batches of ten as this seemed to be a good balance between the number of buttons and the number of elements in the dropdown menu on the page at the time.

Figure 3.2 displays the relations between the different components of the website in a use case diagram. A user can select a test from the given selection after logging in. After answering the test, the server checks for correctness. In case of an incorrect choice, the correct answer is shown. If the user wants to do another test, the website chooses a new random question of the same type.



The Login Page. The Register Page.

Figure 3.1: The login and register pages of the website. A user can log in with their email address and password. The username is used to portray the user on to all other users of the website.

3.2.1 Types of tests

The main types of tests that are used in this website are multiple choice tests, tests where the users have to input an answer and what we call matching test. In the matching tests the users have a block with 5 statements. Additionally, they have another 5 statements in a dropdown where each one only matches with one from the aforementioned block. So they have to find the matching pairs and get them all 5 correct in order to answer the question completely correctly. In cases where some are matched right but not all, it is shown that the given answer is partially correct.

It was tried to have 2 different blocks with the respective 5 statements where the user would have to match in accordance to the number of the statements on the blocks. So for example that the first one of the first block matches with the third one of the second block. Unfortunately while trying to set this up, the statements were always shown in the same sequence. So the first ones always matched, the second ones of both blocks etc.

When it was tried to mix up the order, the matching connection got mixed up too, so even though for example the first one of block one should match the fourth one of block two, the matching connection then got mingled that the first one of block one matched for example the second one of block two. Which would be false but shown as correct. Therefore it was settled to show the second block of statements as a dropdown button for every statement, as this still satisfies the way the test can be done and the criteria of having the order of the answers mixed while keeping the correct connection intact is met.

For each type of task a distinct react component was created which is reused every time that kind of task is accessed by the user. This way the layout stays consistent across the different tests and it reduces overhead as there are only 3 components for the dozens of different tests instead of one component for each one.

As mentioned before, the tests are in a file named moodleX.json and several instances of those tests are within one file. Each instance has slight differences. So they are of the same difficulty and type but they don't have exactly the same arguments in each question. To load the correct test we read the test number from the url which is sent when the user selects the task he wants to do. After loading the correct test, the data is parsed so that the program can read it in a way that is usable. We use the function *Math.random* , which gives us a number between 0 and 1, and multiply it by the number of test instances that are available in the file. That number is rounded down with the function *Math.floor* to get a whole number. That way, each user gets a different randomised question from the ones available and can work out for himself how to solve it. Then the question (with the corresponding answers in case of the matching tests or multiple choice tests) is set and displayed to the user.
The whole process which was explained above can be observed in the code snipped below:

```
useEffect(() => {
  const dataPath = `../jsondata/moodle${testId}.json`;

  fetch(dataPath)
  .then((response) => response.text())
  .then((text) => {
    return JSON.parse(text); // Parse the response text as JSON
  })
  .then((data) => {
    const randomIndex = Math.floor(Math.random()
                  * data.quiz.question.length);
    setQuestion(data.quiz.question[randomIndex]);
```

```
  })
  .catch((error) => setError(error.message));
}, [testId]);
```

A button is also added for the user to get a new question of the same test subject he is already working on. Thus he does not need to leave the exercise page if he wants to continue to deepen his knowledge of a specific exercise type and topic. Figure 3.3 shows an example of how a test page is displayed to the user. It is shown how the page looks like before answering and how it is shown after a user submits an answer. This example demonstrates the appearance of a multiple choice test. The other test types are shown in the appendix.

Another functionality that was added is a button that starts a Timer. That way, the user can try to solve the exercises under time pressure and thus simulate exam conditions, where one does not have infinite time to solve the different questions. As a result, a user can learn to not only solve the exercises without pressure but also solve them efficiently as every second counts. If the timer runs out, the test is automatically submitted as to not lose possible already selected answers. The choice to use this timer is up to the user seeing that he may not want to be under time pressure especially if he is doing an exercise for the first time.

3.2.2 Hints

Hints were implemented in the test where the users have to identify suitable test candidates to eliminate x from specific (in)equations in virtual substitution. The test generator was modified in such a way, that hints are generated at the same time with the questions. So every test question has its corresponding hints when the test file is created.

The first hint is a static text message that briefly explains what the user has to do in case he doesn't know. The second hint does the first step in simplifying the different (in)equations presented to the user. The third hint shows the finalised simplification where solely x is on the one side of the (in)equations. The last hint shows which are the resulting test candidates, as the users may still struggle to remember when an infinitesimal has to be accounted for the solution to be a valid test candidate.

On the test page components, a Hint button is implemented to display the hints to the user if he should need assistance. The user can toggle between the different hints as he sees fit to try to better understand the assignment.

3.3 Backend

For the backend we also use Node.js to implement the functionality of the backend. Together with Node.js, we use Express.js, which is a Node.js library [exp]. It is a convenient web application framework which is helpful for backend purposes. It has an abundance of features which ease the backend development and allow for simple and quick creation of APIs. API stands for application programming interface and in essence it is a way for different computer programs, or parts of a program (i.e. here

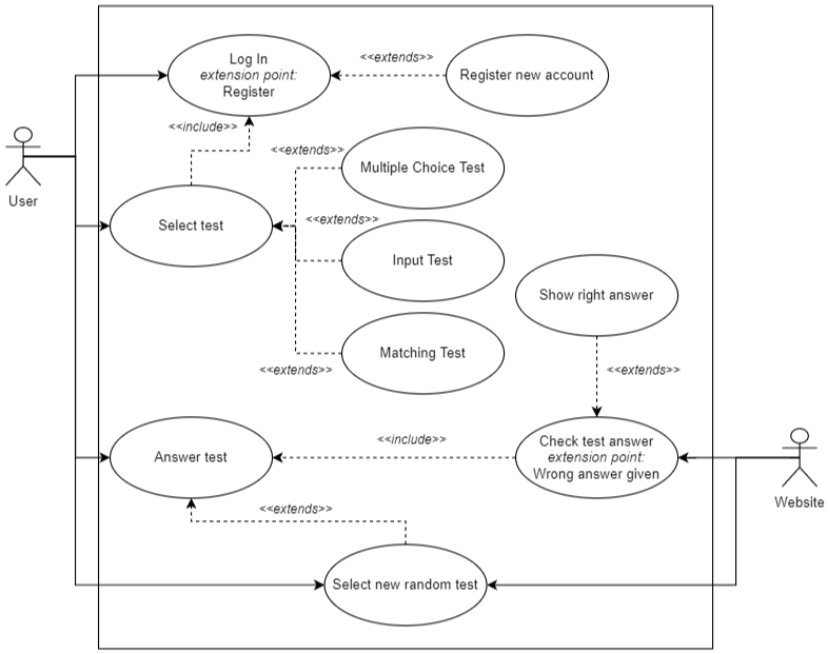


Figure 3.2: A use case diagram of the different components of the website.

backend and frontend of a web page) to communicate with each other. In order to do this, the database is hosted by the server in the backend. A "users" component holds the API functions that allow communication from the frontend with the database through the backend. So when the user tries to sign up in the frontend, the backend writes the user data into the database. To assure safety the password isn't saved as plaintext. The Node.js library bcrypt is used to hash the password to ensure security [bcr].

To provide a secure and efficient way to handle user authentication and authorisation, JSON Web Token (JWT) is used [jwt]. It offers a plethora of qualities that make it a good choice for this task. JWTs allow for stateless authentication, meaning the server does not need to store session information on the server side. This reduces the load on the server and makes it easier to scale the application. Furthermore JWTs are compact and self-contained, they can be efficiently transmitted over the network. JWTs can also have expiration times, which adds an extra layer of security. This ensures that even if a token is intercepted, it will only be valid for a limited time.

3.4 Database

The Database is written in PostgreSQL[pos]. The database consists of 2 tables, a "users" table and a table called "answered_questions". In the users table, the name, email address and the hashed password is saved alongside a user id that every user gets attributed.

In the answered_questions table every answered question of the user is saved alongside the email address to know to whom the answered questions are to be attributed. The questions themselves are saved by memorising the test id of the test that was taken and the specific question in the test file that the user got displayed. To know if it was answered correctly a boolean "answeredCorrect" is also saved. Thus if it is set to false, it is easily identifiable and can be retrieved so the user can try it again if he chooses to attempt his previously incorrectly answered questions. The correctly answered questions can be summed up to place the user on a leaderboard.

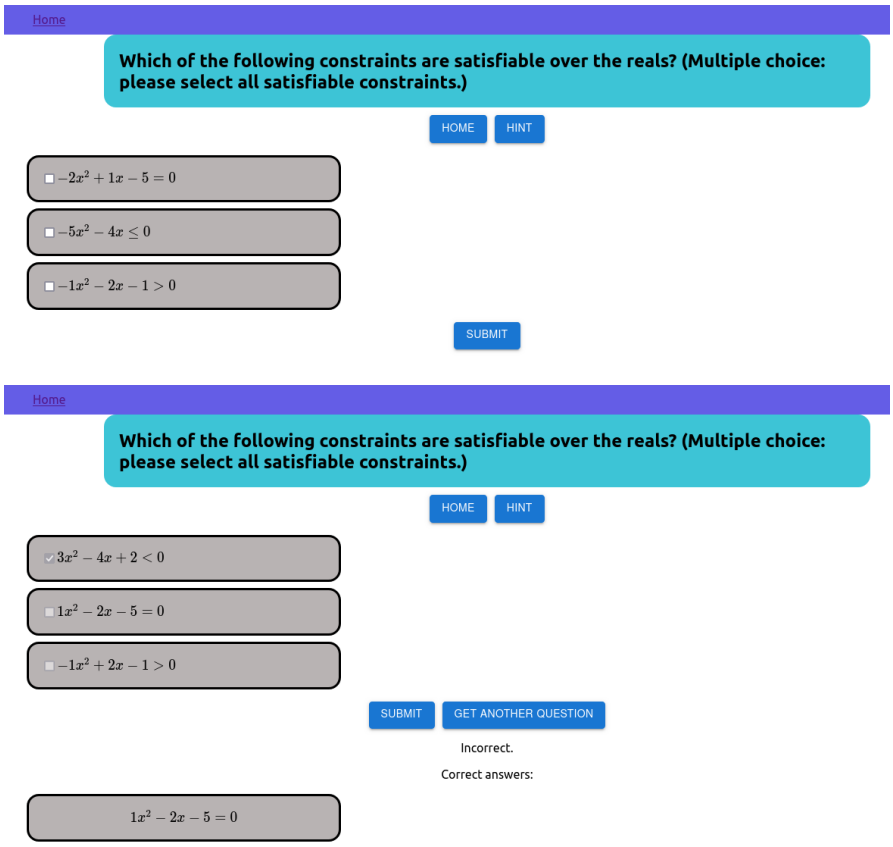


Figure 3.3: *Top*: An example for a test page, in this case for a multiple choice test. The question is shown in a coloured block. Below it, the user has to give the answer. If implemented, the user can let the website give him a hint when he has difficulties with the question. After an answer is logged in, the user can confirm his selection by clicking on the submit button.
Bottom: The same test, after the user has clicked on submit. If the answer is wrong, the right answer will be shown below his selection. If the user clicks on *Get Another Question*, a new question of the current category is selected randomly and loaded.

Chapter 4

Conclusion

4.1 Summary

In this thesis, we have developed a web application to provide users with exercises related to satisfiability checking in theoretical computer science. The application offers various types of exercises, including multiple choice questions, text-input-based questions, and matching tasks.

The web application is designed to engage students in the subject of satisfiability checking by providing them with a user-friendly interface to solve exercises effectively. It offers features like randomised questions, hints for selected exercises, a timer, and the ability to reattempt previously answered questions.

The backend of the application is implemented using Node.js and Express.js, allowing for secure user authentication and authorization using JSON Web Tokens (JWT). The data is stored in a PostgreSQL database, which includes user information, answered questions, and exercise details.

4.2 Future work

This application can be expanded upon in numerous ways. The most apparent way would be to add hints to the other exercise types. Furthermore, a functionality which suggests to the user which tests to practise could be added. Criteria that could be taken into account in implementing that, could be when a certain test was last done or how often a specific exercise was answered incorrectly. Another idea would be to add gamification elements, i.e. a progress bar. This would incite the users to try every different exercise type and would cause them to deepen their knowledge. These are just some ideas in which way this work can be broadened and improved but these are in no way limits on what can be done with this application.

Bibliography

[ÁK16] Erika Ábrahám and Gereon Kremer. Satisfiability checking: Theory and applications. In *Software Engineering and Formal Methods: 14th International Conference, SEFM 2016, Held as Part of STAF 2016, Vienna, Austria, July 4-8, 2016, Proceedings 14*, pages 9–23. Springer, 2016.

[ÁNP23] Erika Ábrahám, Jasper Nalbach, and Valentin Promies. Automated exercise generation for satisfiability checking. In *Formal Methods Teaching Workshop*, pages 1–16. Springer, 2023.

[bcr] Official bcrypt website. `https://www.npmjs.com/package/bcrypt`. Accessed: 2023-11-01.

[BCSV14] Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. A system for generation and visualization of resource-constrained projects. In *CCIA*, pages 237–246, 2014.

[CÁ11] Florian Corzilius and Erika Ábrahám. Virtual substitution for smt-solving. In *International Symposium on Fundamentals of Computation Theory*, pages 360–371. Springer, 2011.

[EE22] Greda Eshiba-Emir. Automated exercise generation. 2022.

[exp] Official express website. `https://expressjs.com/`. Accessed: 2023-11-01.

[Fil] Antoniu-Paul Filip. Automated exercise generation for satisfiability modulo real algebra.

[for] Official formik website. `https://www.npmjs.com/package/formik`. Accessed: 2023-11-01.

[HKR21] Petra Hozzová, Laura Kovács, and Jakob Rath. Automated generation of exam sheets for automated deduction. In *International Conference on Intelligent Computer Mathematics*, pages 185–196. Springer, 2021.

[jwt] Official json web token website. `https://jwt.io/`. Accessed: 2023-11-01.

[Koš16] Marek Košta. New concepts for real quantifier elimination by virtual substitution. 2016.

[noda] Node.js github repository. `https://github.com/nodejs`. Accessed: 2023-11-01.

[nodb] Official node.js website. `https://nodejs.org/en`. Accessed: 2023-11-01.

[npma] npm github repository. `https://github.com/npm`. Accessed: 2023-11-01.

[npmb] Official npm.js website. `https://www.npmjs.com/`. Accessed: 2023-11-01.

[NPM+14] Srinivas Nedunuri, Sailesh Prabhu, Mark Moll, Swarat Chaudhuri, and Lydia E. Kavraki. Smt-based synthesis of integrated task and motion plans from plan outlines. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 655–662, 2014.

[Oh16] Chanseok Oh. *Improving SAT solvers by exploiting empirical characteristics of CDCL*. PhD thesis, New York University, 2016.

[pos] Official postgresql website. `https://www.postgresql.org/`. Accessed: 2023-11-01.

[reaa] React documentation. `https://react.dev/`. Accessed: 2023-10-27.

[reab] React github repository. `https://github.com/facebook/react`. Accessed: 2023-11-01.

[Sin07] Carsten Sinz. Visualizing sat instances and runs of the dpll algorithm. *Journal of Automated Reasoning*, 39:219–243, 2007.

[Tin02] Cesare Tinelli. A dpll-based calculus for ground satisfiability modulo theories. In *European Workshop on Logics in Artificial Intelligence*, pages 308–319. Springer, 2002.

[VWM15] Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015.

[Wal14] Johannes Waldmann. Automated exercises for constraint programming. In *WLP/WFLP*, pages 66–80, 2014.

[yup] Official yup website. `https://www.npmjs.com/package/yup`. Accessed: 2023-11-01.

Appendix

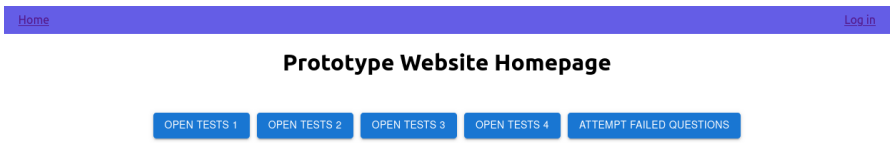


Figure 1: The main page of the website. Here, tests can be selected. It also offers a button to redo previously failed tests.

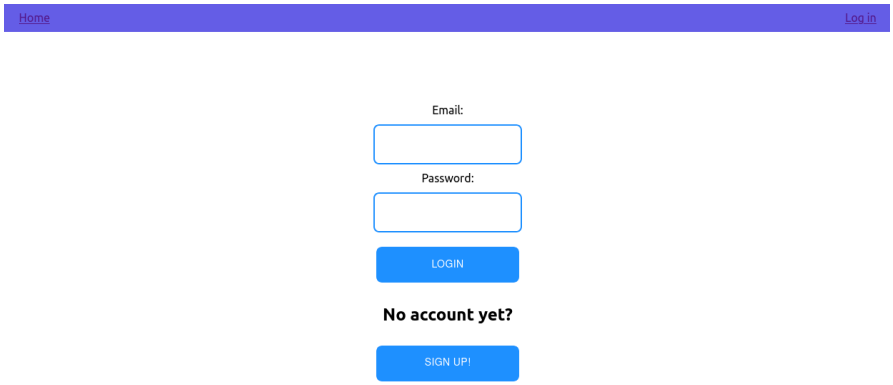


Figure 2: The login page of the website.

Figure 3: The register page of the website. Aside from the email and password that are used for the login, each user can also decide a username.

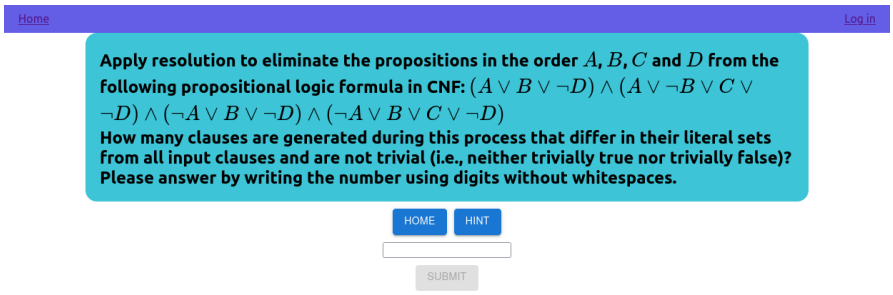


Figure 4: An example for a test with a text field as input. The user must give the answer as plain text according to the directions given in the task description.

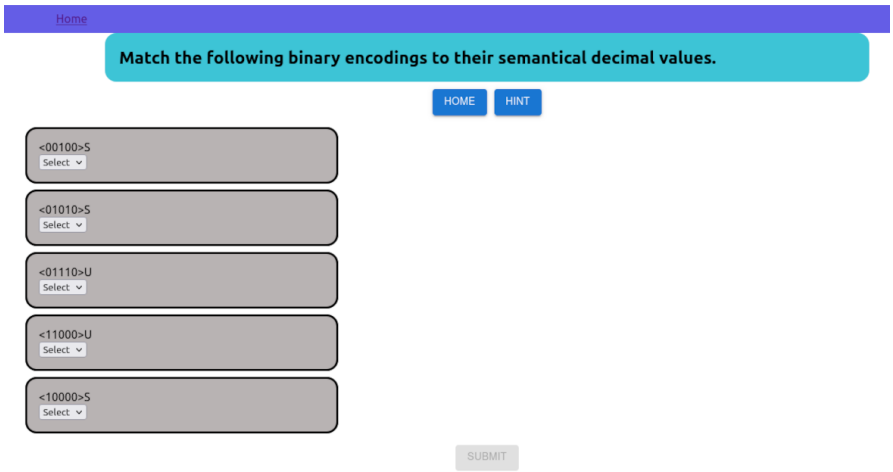


Figure 5: An example for a matching test. For each field shown, the user must select the appropriate answer from a list of possibilities.

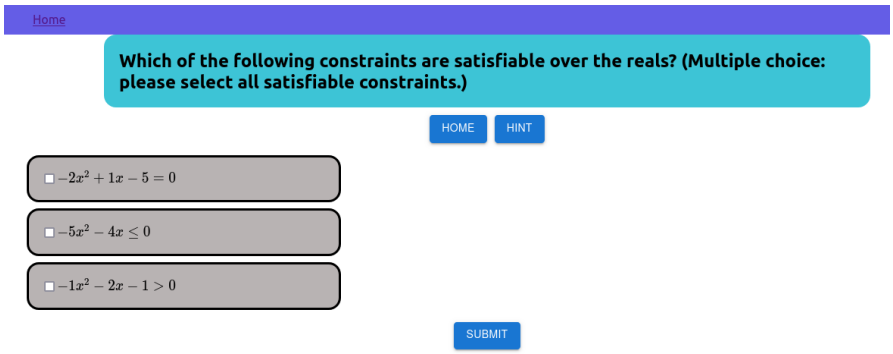


Figure 6: An example for a multiple choice test. The user has to select all right choices. It can be zero, requiring no selection.

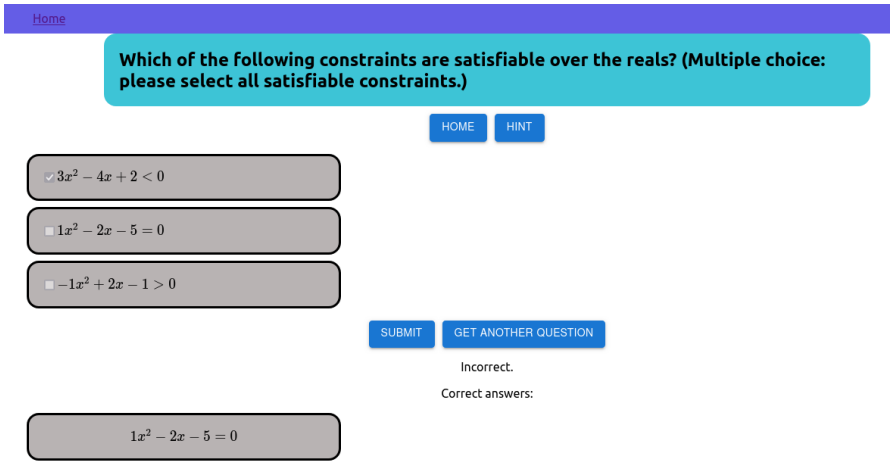


Figure 7: An example for a test – in this case multiple choice – showing the right answers if the answer of the user was wrong.