NoDD Algorithm Implementation

Andrew Plum & Nathan Nguyen

Department of Computer Science, University of Idaho

plum0598@vandals.uidaho.edu

nguy7866@vandals.uidaho.edu

1 Introduction

We were tasked with the creating of an evaluation system with the ability to grade student submitted proofs involving functional dependencies. This project is the next step in the creation of a tutoring system concerning concepts in normalization and database design theory. The majority of our project was developed on the backend. Our work is to be integrated with the existing work of previous projects to reach a complete tutoring and evaluation system. The grading system we created utilizes set definitions of functional dependencies. To understand our methodology, it is required to delve into the theory database design and normalization.

2 Background and Theory

2.1 Functional Dependencies

In relational database theory, functional dependencies serve as the basis for defining relationships between attributes within a database. A relation in a relational database consists of a set of attributes and a table of tuple-rows i.e. a set of tuples (or rows) where each tuple contains an attribute-value pair for all attributes in the relation. We say two sets of attributes X,Y are functionally dependant X→Y if for every instance of the same X attribute values that occurs within the table, the value of the Y attributes are not distinct. In other words the values of the attributes in X determine the values of the attributes in Y, and Y depends on X. A relation scheme is defined to be a relation paired with a set of functional dependencies. Every relational database is characterized by its underlying relation scheme.

As an example, Table 1: Math 101 Students illustrates a basic functional dependency where "Student ID → First Name, Last Name, Age" holds. Note that "Last Name → Student ID" also happens to hold, however if more students were added to the table, this dependency may not continue to be enforced, while the Student ID most likely will continue to discriminate students and determine the other attributes. Also note as a counter example, that "First Name ∤ Age" since the name "Alice" corresponds to two distinct age values, 19 and 20.

The relation scheme for this table might rightly be given as ⟨ R({`Student ID`,`First Name`, `Last Name` `Age`}),

F={`Student ID→{`First Name`,`Last Name`,`Age`}`⟩

where R is the relation and F is the set of functional dependencies.

2.2 Armstrong's Axioms

As part of the formal definitions for functional dependencies on a relation, Armstrong's Axioms or inference rules, provide a logical framework for finding all the functional dependencies of a relational database. There are three primary rules or axioms, and several secondary rules which can be derived from those axioms.

Applying the set definitions of functional dependencies plays an integral role in our evaluation system for the logical consequence of F and attribute closure. This is because the set definitions are used as rules which are referenced in each of the steps in the proofs. The set definitions that are used as rules are listed here:

1. Reflexivity: If X, Y ⊆ U and Y ⊆ X, then X ⊆ Y holds.

2. Augmentation: If X → Y and Z ⊆ W ⊆ U , then XW → YZ.

3. Transitivity: If X → Y and Y → Z, then X → Z.

4. Union Rule: If X → Y and X → Z hold, then X → YZ holds. Recall YZ = Y ∪ Z.

5. Decomposition Rule: If X → YZ holds, then X → Y and X → Z hold.

6. Pseudo-transitivity Rule: If X → Y and YZ → W hold, then XZ → W holds.

2.3 Proofs Involving Functional Dependencies

Various proofs involving functional dependencies can be done all with the goal of normalizing the database. In the tutoring and evaluation system being designed, students will submit these aforementioned proofs so that their understanding of concepts in normalization and database design can be evaluated and guided towards improvement when necessary. The proofs we were tasked with grading were proofs for the logical consequence of F and attribute closure.

2.4 Proofs for the Logical Consequence of F

The logical consequence of F, where F denotes a set of functional dependencies, encompasses all of the functional dependencies which can logically be derived from F. An example of a logical consequence of F would be where F is the set {A→B, B→C} and where we say A→C is logical consequence of F by transitivity between the first and second given functional dependencies in F. To see if a functional dependency is a logical consequence of F, a proof can be performed where each step in the proof references a rule to demonstrate that the new functional dependency is a logical consequence of F. If F is a set of functional dependencies, the logical consequence of F involves finding all of the functional dependencies that logically follow from F, F\*, which means that the attribute closure of F and F\* should be the same.

2.5 Proofs for Attribute Closure

Attribute closure is all the possible attributes which can be derived, X+, when given an initial set of attributes, X, and set of functional dependencies, F. X+ is found by repeatedly applying set of attributes you currently have to the set of functional dependencies until no more attributes can be added to the current attribute set. Relating this to the previous example, if given X={A} and F={A→B, B→C}, the attribute closure X+ would be {A, B, C} as it encompasses all the attributes implicated by the given functional dependencies. A proof can be performed to derive X+ where each step references what new attributes can be derived while referencing the functional dependency in set F in which it was derived.

3 Methodology

3.1 Programming Languages and Tools

Since our project was mainly focused on backend operations as well as storing and retrieving data from the database we designed, we decided to use the programming languages PHP and SQL. We also utilized XAMPP, PhpMyAdmin, and GitHub in the development of this project.

3.2 Notepad++ and Visual Studio Code

Notepad++ had the advantage of being a lightweight text editor; it offers syntax highlighting and various plugins for coding to streamline the development process. Visual Studio Code on the other hand has even more robust extensions and features to offer than Notepad++ and was used for as an environment for debugging as well as coding. These tools in combination facilitated the coding development process due to their different strengths.

3.3 Git and GitHub

Because this was a collaborative team project, we needed some form of version control for the code we developed. We chose to leverage Git through GitHub. GitHub has the benefit of having an easy user-friendly graphical interface to utilize Git. GitHub allowed us to track and manage the changes we made to the code as well as reverting changes to previous versions if needed. For collaborative coding, the pull request and branching features were invaluable for our project as it allowed us to work on separate features simultaneously and merge the contributions seamlessly. It also encourages the development of more modular programming, which is always a desired feature in one’s code. These features all ensured a more organized and cohesive codebase. We also utilized GitHub’s feature to host our code in a repository there.

3.4 XAMPP

XAMPP, a cross-platform web server solution, was used for testing and debugging of PHP applications in its local development environment. Because it consists of Apache, MySQL, PHP, and Perl, XAMPP simplified the setup of a complete web server stack on our local machines, and was chosen for its straightforward installation to facilitate a consistent development experience and compatibility between team members and the products.

3.5 PHP

Hypertext Preprocessor (also known as PHP for short) is a server-side scripting language that is very similar to C in its implementation. It allows for dynamic content generation and clean integration with web applications on the frontend and databases on the backend, which was a required feature for the programming language we needed for this project. We could have developed the backend of this project in python, but ultimately chose PHP because most of the preexisting tutoring projects already used PHP so we decided to be consistent with this design choice; it also had the added benefit of being a fun programming language that we had never used before.

3.6 SQL

For database management and manipulation Structured Query Language (SQL for short). SQL allows for efficient retrieval, insertion, modification, and deletion of data in the database; and the specific distribution of SQL we used for this project was MySQL. The language’s standardized and natural English like syntax in conjunction with its relational database capabilities all helped in the successful development of the database for the project.

3.7 PhpMyAdmin

PhpMyAdmin was utilized because it offers a graphical user interface for MySQL within the XAMPP environment, making database administration more straightforward and efficient. Its simplified features help us in performing tasks such as executing SQL queries, creating tables, and managing relationships between and within tables. PhpMyAdmin enhanced our ability to understand the underlying data structure of the database because of its ability to visualize and manipulate the database schema.

3.8 Grading and Feedback

We decided to start simple in the grading scheme where a student proof submission had to state after being graded: it was correct, or it was incorrect. How we would determine if the student’s proof was correct would be if each of the steps of the proof was valid and then if the final step reached the desired correct answer the proof was trying to prove. In checking if each of the steps of the proof are valid, we would check in the sequential order in which they were written. If all the steps in the proof were determined to be valid, then we would determine if the desired correct answer was reached in the last step. If this was true, then the students’ proof would be graded as correct. And if any of the checks we performed on each of the steps of the proof failed, then we would stop where the first check failed, and the students’ proof would be graded as incorrect. In addition to indicating that the students proof was incorrect, we would return additional feedback of what step the specific check failed in the evaluation process. The additional feedback returned did not include whether every step after the failed check was correct because those steps may be considered as correct by the grading system we designed even though they were referencing an incorrect step.

3.9 Specific Grading Cases

Steps that give what could be a correct functional dependency but have an incorrect justification are marked as incorrect. Also, because unnecessary steps (steps which are not referenced by future steps) under the current grading system are marked as valid, they will still lead to the proof being graded as correct even though the proof is not concise. We decided to make this the case because unconcise proofs are still considered just as correct of a proof as a concise proof. One other thing to note is we did not have to worry about marking steps that would be valid if preceded by the correct steps as incorrect if out of order and not preceded by the correct steps because there would be no way the student could reference the previous steps if they did not already write out those previous steps to be referenced; this was enforced by the nature of the interface the student is working with when writing their proof. If the student did attempt to write a correct proof by writing what is a correct future step, then the step would be marked wrong because it has an incorrect justification because it could not be referencing the correct preceding steps that the student hasn’t yet written.

4 Binary Implementation

We had to figure out how to represent the functional dependencies so that the set definitions could be applied to each of them and their derived functional dependency in each step of the proof. We wanted to keep the way that the functional dependencies were when we were working with them and how they were when we stored them in the database consistent with each other. Noticing that each attribute in each functional dependency had to state – it was either present in its respective side of the functional dependency or it was not present – we decided to represent each of the functional dependencies in two binary numbers. So, if we have the relation ABCDE and we have the functional dependency AB→E, this functional dependency would be represented by binary number 11000 for its left side and 00001 for its right side. Similarly, if we have the functional dependency ACE→BDE, the left side of it would be represented by 10101 and the right side of it would be represented by 01011. Because we were storing the functional dependencies in binary rather than some other sort of data representation like a string, there was the ability to easily convert to and work with numbers in decimal and hexadecimal. Storing the functional dependencies in binary also had the advantage of being able to work with all the features and operations of numbers in the programming language we were working with.

4.1 Binary Operators

What was particularly useful about the binary representation of the functional dependencies is that we could use the binary operators when applying the set definitions. For example, if you have set X and set Y, X can be described as a subset of Y if the AND between X and Y is equal to X. And because X and Y are represented in binary in our grading system, we can simply use the built-in AND operator (in PHP it is “&”) to apply the AND (so it would be written “X & Y = X”). So if set X={B} and Y={ABC} then it would be determined X is a subset of Y but Y is not a subset of X. X would be a subset of Y because X in binary is 010 and Y in binary is 111 and applying the rule accordingly 010 & 111 = 010. As another example, the difference between sets X and Y can be represented by the XOR operator (in PHP it is “\”). Determining the difference between sets can be useful in determining if a functional dependency is derived by decomposition from another functional dependency. By using the set operators, we can easily apply all 6 of the set definitions when checking if a functional dependency can be derived through one of the definitions.

4.2 How the Binary Operators are Applied in Each of the Set Definitions

5 Database Design Implementation

The schema for the noddx database is designed to support the storage and organization of various types of data related to submissions, questions, assignments, and feedback. The database consists of six tables, each with a specific purpose and relationships with other tables.

The noddx\_entries01 table contains information related to individual entries for questions concerning the logical consequence of F. The information in this table includes is an id to uniquely identify each entry, submission\_id to link to the submission it belongs to, numline to denote the line number of each entry of the students proof based answer, sideleft and sideright to specify the left and right side decimal values where their binary equivalent represent functional dependencies the student is using in their entry, and rule to identify the rule associated with the entry such as “Given” or one of Armstrong’s Axioms. Each rule is encoded as an integer. Additionally, the table includes a column called refs which stores textual references which correspond to the numlines of other entries in table noddx\_entries01 that are a part of the same submission as the current entry or entries in table noddx\_qentries which is initial given entries that are a part of the same submission. The entries in this table can contain either zero, one, or two references where for zero references it is blank, for one reference it is “X”, and for two references it is “X, Y” where X and Y are integers in strings which are referring to numlines for entries in the proof either for noddx\_entries01 or noddx\_qentries.

The entries in the table noddx\_qentries represent the initial information that the student is given to create their proof when responding to the question asked. The noddx\_qentries table is similar to noddx\_entries01 but is specifically for question entries. It includes an id to uniquely identify each entry, question\_id to link to the corresponding question, numline to denote the line number, sideleft and sideright to specify the left and right side decimal values where their binary equivalent represent functional dependencies.

The noddx\_submissions table is designed to capture submission-related data, with fields for id, timestamp for the time the submission was made, question\_id to link the submission to one of the specific questions, user\_id to link to the user who made the submission, and feedback to store textual feedback for the submission which will be displayed to the user after their submission has been graded.

The noddx\_questions table contains information about questions, including an ID to uniquely identify each question, assignment\_id to link to the assignment it belongs to, name to store the name of the question, type to specify if the question is a logical consequence of F question or an attribute closure question, and data1 and data2 to store additional data related to the questions where data1 is the final left side of the functional dependency which should be proved for the logical consequence questions or it is the initial set for the attribute closure question and data2 is the final right side of the functional dependency which should be proved for the attribute closure questions.

The noddx\_entries02 table is similar to noddx\_entries01 but with the attribute list instead of sideleft and sideright in the noddx\_entries01 table; list should be a decimal value whose binary equivalent represents the current set of attributes in our attribute list. The table like the noddx\_entries01 table also has the attributes submission\_id, numline, rule, and refs all with very similar implementations. It should be noted that any rule can be inputted for the attribute rule but “Given” and “Reflexivity” are correct for the first entry of the submission and “Given” or “Null” (nothing) should be used as the rule for every entry after the first entry in the same submission. The attribute refs should always have one reference which functions identically to the reference in noddx\_entries01.

Lastly, the noddx\_assignments table contains information about assignments, including an id to uniquely identify each assignment and a name to store the name of the assignment.

The schema makes use of primary and foreign key constraints to establish relationships between tables which helps in ensuring data integrity and facilitating efficient querying. The use of integer data types for IDs and references to other entities provides a standardized and efficient method for uniquely identifying and linking records. Also, the inclusion of text data types allows for flexibility in storing textual information such as feedback and references.

The schema we designed is supposed to provide a complete and structured framework for storing and managing the diverse data elements related to entries, submissions, questions, and assignments in the noddx database.

6 Display Design Features

Our project was primarily developed on the backend. However, we still needed an effective way to display the results of the student submissions being graded by our algorithm.

This is a screenshot of the display of our web page where the user has submitted their responses to a question and your algorithm has graded their work. At the top of the page, there is a field labeled "Submission ID to check," indicating that the system allows users to enter a submission ID to view the graded results. Below that, the specific question in focus is displayed: "Using the Logical Consequences of F: {A->B, B->C, A->D, CD->AE}, Show that BD->AC."

The bulk of the page is then divided into two sections: "Student entries" and "Justification."

In the "Student entries" section, there is a table indicating the steps the student took to derive the conclusion "BD=AC". For example, it lists entries such as "Pseudo Transitivity 2,4," "Augmentation 2," and "Decomposition 7." Each entry has a checkbox indicating whether it is correct or not.

In the "Justification" section, the displayed result is "Correct!" and there is a line showing the logical steps taken to derive "BD=AC," specifically "BD=ACE Union 5,6" leading to "BD=AC."

Finally, at the bottom of the page, there is a row labeled "Valid?" with checkboxes indicating the validity of some of the steps made by the student.

Overall, this page provides a detailed breakdown of the student's submission and the steps they took to derive their answer, along with a clear justification and assessment of the correctness of their work.

OR

The display for the logical consequence of F questions showcases the evaluation results for student answers in the format of the question prompt, student submissions, and the resulting evaluation. Each submission is labeled with a unique Submission#X (where X is the submission ID), followed by the individual Entry, Justification, and Validity status. The results are broken down into the Given portion, which lists the initially given information, and the Student Entries, which represent the student's responses. The final evaluation verdict for each submission is provided under the Result section along with any extra feedback we could return to the student.

As for the attribute closure questions, the display structure follows a similar format, with the Submission#X (where X is the submission ID) label, Entry, Justification, and Validity sections. The Given portion outlines the initial information presented, and the Student Entries showcase the student's input. The evaluation result of whether the student is correct or not, and any extra descriptive information we could return, is presented under the Result section.

Let’s look at some example results of a submission to a logical consequence of F question after the submission has been evaluated.

AND

The student's answer and the logical steps to prove BD->AC are displayed on the webpage. The submission has been marked as "Correct," indicating that the algorithm has recognized the student's response as accurate. The student's logical steps to prove BD->AC are as follows:

1. B->C Given 2
2. CD->AE Given 4
3. CD->ACE Augmentation 6
4. BD->ACE Pseudo Transitivity 5, 7
5. BD->AC Decomposition 8

In addition, there is a "Valid?" column showing "✓" for each step, confirming the correctness of the logical steps taken by the student. Overall, the evaluation algorithm has successfully verified the student's submission and provided the logical steps needed to prove the given functional dependency.

AND

In this second example of a student submission, we can see the student has provided their answer and justification, and the evaluation algorithm has determined the result to be an incorrect response with a problem in the logic. The student entry for attribute closure in the given functional dependency has been evaluated step by step, and the evaluation result shows where the student's logic has gone wrong. Based on the evaluation, the attribute closure for the final attribute set by the student seems to have been incorrectly calculated, as indicated by the checkmark under "Justification Valid?" and the "X" in the "Result" column. The evaluation algorithm has identified the problem in the logic, as indicated by the note "Incorrect. There's a problem with the logic." which makes sense since the “Given 4” justification corresponds to the 4th entry which contains the functional dependency “CF->B” which can not be used to add F to the current attribute set. It should be noted that no more of the proof is displayed after an incorrect step as it is difficult to determine if the succeeding proof steps are impacted by the incorrect step, and ultimately there is the same result of an incorrect proof. If the student wants to see if the steps after an incorrect step are correct, they can fix the incorrect step in their proof and reevaluate their answer.

AND

Both displays effectively organize the student submissions, the corresponding justifications and validity assessments of each submission, and the final evaluation result of each question, providing a clear and structured overview of the evaluated proofs for each type of question.

7 Testing and Evaluation

From all the testing we have performed, we are confident that our algorithms will perform as desired. For one, we believe that our application of the underlying theory is sound. To be sure of it, we have performed extensive testing. We tried to perform a variety of tests where our answers highlighted different applications of the theory in order to catch as many edge cases as we could. Some of these tests have included cases provided by our instructor as well as cases designed by us. If you are not yet confident of our algorithm implementation, it is easy to create and perform your own tests. To do so the data should first be entered into the database, and then on the web page interface, you can perform the tests of the cases entered the database. From there, you can visually evaluate if our results align with what they should actually be.

8 Future Improvements

Future improvements to the project could focus on enhancing the grading and feedback mechanisms to provide a more comprehensive and personalized learning experience for students. In addition, the project could be expanded so that more question types could be graded. Here are the key enhancements that could be made explained in more detail:

8.1 Better Grading

A more detailed oriented grading system could be implemented that goes beyond the current binary correct/incorrect assessment. The system should allow teachers to define grading criteria on a scale, such as out of 10, and take into account factors like the number of correct steps versus incorrect steps made by the student. The goal would be to provide a more accurate and fair evaluation of the student's understanding and performance.

8.2 Better Feedback

8.2.1 Personalized Feedback

More detailed feedback on incorrect student answers can be provided, including specific explanations of why the answer is wrong. This personalized feedback would help students understand their mistakes and learn from them, ultimately improving their grasp of the subject matter.

8.2.2 Complete Feedback

Currently, if a step in a student's proof is wrong, the evaluation system does not grade after the incorrect step. The evaluation algorithm could be improved so that it grades every answer after the first incorrect step. A design decision would need to be made on whether to mark a subsequent step which references and incorrect step as correct because the logic it uses is correct even though it uses an incorrect step as a reference.

8.3 Hinting System

Another useful feature that could be implemented is a hinting system which guides students towards the correct answer. This feature should be customizable, allowing instructors to enable or disable it based on their teaching approach. The hints might also alter the way the assignment is graded if used by the student; this feature would again ideally have the ability to be enabled or disabled by the instructor.

8.4 Grading More Question Types

8.4.1 Candidate Keys

In the future, the evaluation algorithm can be expanded to include the grading of student proofs for finding candidate keys. The kinds of candidate key proofs that could be graded include the exhaustive, heuristic, or tree method. This would provide a more comprehensive evaluation of the students' understanding of the concept.

8.4.2 Normal Form Decomposition

Another potential improvement that can be made to the project is the introduction of a feature to check if the student correctly decomposed a database schema into the third normal form and Boyce-Codd normal form. Because checking if the student performed the normal form decomposition correctly requires finding a canonical cover of the initial set of functional dependencies and the appropriate candidate keys that correspond to it, the functionality of this feature likely involves the functionality of being able to check if a functional dependency is a logical consequence of a set of functional dependencies, being able to find the correct attribute closure given a set of functional dependencies and an initial set of attributes, and being able to find all of the correct candidate keys of a set of functional dependencies. Implementing this feature to the evaluation system would provide a more thorough assessment of the students' grasp of database normalization concepts.

Conclusion

My partner and I take pride in the accomplishments we have achieved. Our innovative approach to creating evaluation algorithms for the logical consequence of \(F\) and attribute closure required conscientious thought and extensive brainstorming. Along with algorithm implementation, our design and successful querying of a tailored database showcased our technical proficiency. The binary implementation of our work was notable as it will save future groups of the time we undertook in the creative design process. While we acknowledge there is always room for improvement with our project, we have provided a project that future contributors can easily build upon; it will be interesting to see how they expand upon it. Ultimately, we think our project stands as a testament to our learning this semester.