THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

SQL EXTENSIONS FOR DOMAIN AGNOSTIC DATA REPRESENTATIONAL CONSISTENCY

Bingyu Yi

Bachelor of Software Engineering

A thesis submitted for the degree of Master of Philosophy at

The University of Queensland in 2016

School of Information Technology & Electrical Engineering

Abstract

The explosion of big data exacerbated the significance of data quality in decision support systems and data warehouses. Data inconsistency, as a significant data quality problem especially for heterogeneous databases, is mainly researched in three aspects: data integrity, semantics, and representational inconsistencies. The data integrity aspect has been well researched and implemented into DBMSs and data warehouses. The methods to detect and resolve semantic and representational inconsistency problems have been developed within a certain context. However, for a general data quality context, there is a lack of methods available for domain agnostic data inconsistency problems. Historically, data representational inconsistency has already been discussed in the pre-processing of data cleansing frameworks and data quality tools. However, since they deal with the problems in a certain context or based on a specific domain, users must obtain specific information about the data such as the master data and the data dependencies in order to address these data inconsistency issues.

This thesis focuses on domain agnostic data representational inconsistency problems in a general data quality context in a relational database. In this thesis, we employ a declarative method which introduces SQL extensions instead of writing massive amounts of code. To improve data representational consistency, we propose a user-driven pattern-based framework using the iterative and interactive approach and string pattern matching technology. There are three main subtasks: a) design a complete and nearly mutually exclusive pattern library, b) detect all the possible patterns for each record in the target column, and c) unify the inconsistent data records. Then, we improve the pattern detection algorithms for inconsistent data records through a modified DFA (Deterministic Finite Automaton) and comprehensive experiments are conducted to verify the accuracy and efficiency of the proposed approaches. The evaluation results demonstrated that the proposed methods in this thesis have better performance over the naive solution. Finally, we implement a toolkit based on the proposed framework and methods.

Declaration by Author

Publications during candidature

Conference papers:

- B. Yi, W. Hua, and S. Sadiq. A Pattern-Based Framework for Addressing Data Representational Inconsistency. In *ADC*, 2016. (Accepted on 15/07/2016)

Publications included in this thesis

B. Yi, W. Hua, and S. Sadiq. A Pattern-Based Framework for Addressing Data Representational Inconsistency. *ADC*, 2016. - incorporated as Chapter 5.

Contributor	Statement of contribution
Bingyu Yi (Candidate)	Designed algorithms(100%) Designed experiments (80%) Wrote the paper (80%)
Wen Hua	Designed experiments (10%) Wrote the paper (10%) Discussion and analysis of the algorithm design
Shazia Sadiq	Designed experiments (10%) Wrote the paper (10%) Discussion and analysis of the algorithm design

Contributions by others to the thesis

For all the research work included in this thesis, Prof. Xiaofang Zhou, as my principle advisor, has provided very helpful insight in the overall ideas. My associate advisor, Shazia Sadiq, has also assisted with both the refinement of the idea and the technical details.

Statement of parts of the thesis submitted to qualify for the award of another degree

None.

Acknowledgments

This thesis represents a major portion of the research undertaken during my MPhil candidature at the University of Queensland. During these two and half years, I have received great support from my family, friends and the fellows at DKE.

First I would like to thank Prof. Xiaofang Zhou, my principal advisor, who offered valuable guidance on research directions throughout my whole candidature. His insightful comments inspired me to pursue better solutions for each research problem. I would also like to thank my associate advisor, Prof. Shazia Sadiq, for her precious guidance and patient discussion in my research s. Dr Hongzhi Yin, Dr Hailong Liu and Dr Wen Hua also gave me a lot of helpful comments and suggestions during the work of my project and I really appreciate their help and kindness.

My deepest thanks go to my friends at DKE as well. They have helped me so much especially when I was anxious. We shared so many valuable moments.

Last but not least, I owe the most gratitude to my family. Their support is always the power driving me forward, and their love is the most precious treasure in my life. Without their constant understanding and help, I believe I would not have reached this far in both life and learning today.

Keywords

Data Quality, Data Representational Consistency, SQL Extension, Regular Expression, Pattern Detection

Australian and New Zealand Standard Research Classifications (ANZSRC)

ANZSRC code: 080604, Database Management, 80%

ANZSRC code: 080309, Software Engineering, 20%

Fields of Research (FoR) Classification

FoR code: 0806, Information Systems 80%

FoR code: 0803, Computer Software 20%

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data quality is not a new concept which appears along with the data [44]. "Fit for Use" is a judgment of the quality of the data [29]. As a significant dimension of data quality, data consistency problems arise everywhere, whether on a large scale for a company or a country, or a smaller scale for individuals. Data is generated and represented differently in different cultures, countries, companies and contexts using different standards and formats. However, when this disparate data is combined, data inconsistencies become evident such as having different "date" and "name" formats. These inconsistencies can lead to people misunderstanding the data (e.g."04/11/2016" and "11/04/2016" can represent the same date but different formats in American style and Australian style respectively), and have negative effects in data analysis results which may even result in losses in companies [10]. These problems are receiving increasing attention from both industry and academia with data becoming more and more massive in size and heterogeneous [44].

1.1 Background

Data consistency was defined as "format and definitional uniformity within and across all comparable datasets" by Ballou and Pazer [7]. Recently, efforts have been made to improve data consistency. In the industrial and business community, companies such as IBM [1] and SAS [2] provide

[1]http://www-01.ibm.com/support/docview.wss?uid=swg27008803

[2]http://support.sas.com/documentation/cdl/en/etlug/66819/HTML/default/viewer.htm#etlugwhatsnew4.htm

solutions for data inconsistency problems when conducting data integration. Furthermore, some data quality tools such as address standardization tools [3] focus on data inconsistency problems in particular fields. In academia, concepts, frameworks, and methods for data consistency have been discussed. Roger and Paul [13] look at three aspects of data consistency issues: data integrity, semantics, and representational consistency. Data integrity consistency has been well researched with a focus on integrity constraints especially in relational models and has been supported by most DBMSs (Data Base Management Systems) [44]. Approaches such as the rule based method and CFD (Conditional Functional Dependency) [17, 22, 23] are used to improve the semantics aspect of data consistency. In addition, data representational consistency appears in the majority of data cleansing frameworks [37, 43] and becomes a pre-processing task in the process of resolving other data quality problems such as record linkage, data fusion, and entity resolution.

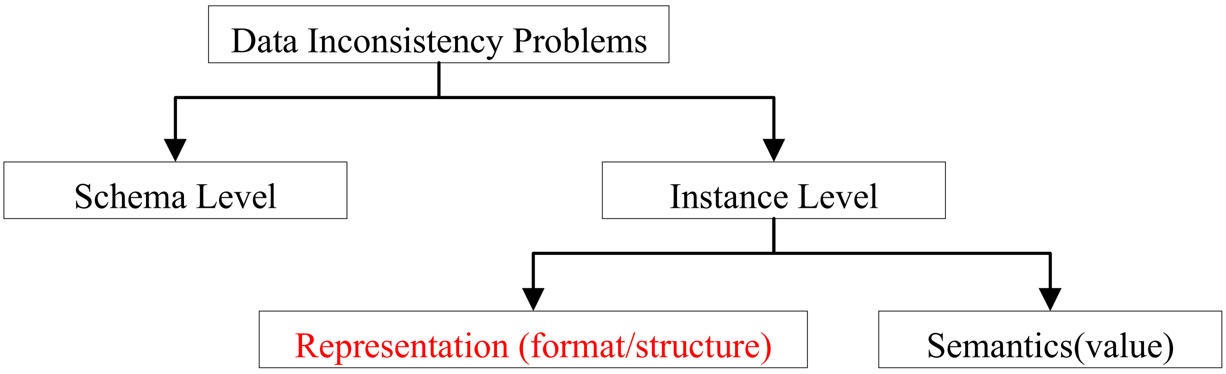

Data Inconsistency Problems
Schema Level
Instance Level
Representation (format/structure)
Semantics(value)

FIGURE 1.1: Data inconsistency problems.

According to [40, 41],we divide these inconsistent representation problems in relational databases into two levels:

1. Schema level: At the schema level, this problem could be conflicting schema across datasets or different information sources and, in particular, different schema representations of the same object in different tables. For example, an address could be represented in one field or decomposed into the fields of street, Suburb and state.

2. Instance level: At the instance level, the problem could be the non-standardized data in one

[3]https://appexchange.salesforce.com/listingDetail?listingId=a0N30000001SqRFEA0

TABLE 1.1: Example 1.

Name	**Street**	**Suburb**	**State**	**Sex**
John Smith	222 Carmody Rd	St Lucia	Queensland	1
Smith, Kate	Carmody Rd 222	St Lucia	QLD	0

TABLE 1.2: Example 2.

LastName	**FirstName**	**Gender**	**Address**
Smith	John	M	222 Carmody Rd, St Lucia, QLD
Smith	Kate	F	222 Carmody Rd, St Lucia, Queensland

dataset or from the same information source, such as having different formats for the same information or using different measurement units.

The two data sets in the example of Table 1.1 and Table 1.2 are both in relational format but exhibit schema and instance data conflicts. At the schema level, there are structural conflicts (different representations for names and address) and name value conflicts (e.g. Sex/Gender). At the instance level, there are different gender representations (“0/1” cf. “F/M”) and different formats (“John Smith” cf. “Smith, Kate”) in the same column. Table 1.3 shows a detailed analysis of the data inconsistency problems in Table 1.1 and 1.2.

In this thesis, we mainly focus on data representational consistency at an instance level in relational databases. Specifically, in Table 1.3, we target the problems of different formats and structures and aim for instance level representational consistency (e.g. “John Smith” $\rightarrow$ “Smith, John”; “Carmody Rd 222” $\rightarrow$ ‘222 Carmody Rd”).

1.2 Motivation

With the increasing amount of data used in business and scientific domains, data quality is of great interest. As one of the fundamental dimensions of data quality, there is no doubt that data consistency is a significant issue not only in business and industry but also in fields of scientific research. Lee et al. [33] maintain that many companies face a multitude of inconsistencies in data definitions, data formats and data values, which lead to difficulties in understanding and using data.

TABLE 1.3: Data consistency problems in Table 1.1 and Table 1.2.

Classification	**Problem**	**Dirty Data**
Schema-Structure	different schema structure	emp1 = {Name}; emp2 = {LastName, FirstName}
Schema-Structure	different schema structure	emp1 = {Street, City, State}; emp2 = {Address}
Schema-Value	different schema value	emp1 = {Sex}; emp2 = {Gender}
Instance-Value	different value	emp1 = {Sex: 0/1}; emp2 = {Gender: M/F}
Instance-Structure	different formats	emp1 = {John Smith}; emp2 = {Smith, Kate}
Instance-Structure	different structure	emp1 = {222 Carmody Rd}; emp2 = {Carmody Rd 222}
Instance-Value	different abbreviations	emp1 = {Queensland}; emp2 = {QLD}

These data inconsistencies are mainly caused by data fusion and data integration processes, the lack of normative data management processes and the need to obtain data from different sources. Recently, efforts have been made to improve data consistency. In scientific fields, inconsistent data will affect analysis results of experiments. In addition, improving data consistency is considered to be a pre-processing task in the process of resolving other data quality issues such as data integration, and it appears in the majority of data cleaning frameworks. As the quantity of data explodes and data inconsistency becomes more pervasive, the need to improve data consistency intensifies.

Current research about instance level representational inconsistency problems generally target a certain domain in a specific context, which we call domain specific. For example, in the healthcare sector there is a need to exchange patient health related data among healthcare professionals and institutions and research into this area is domain specific. Specifically, Churches et al. [18] propose utilizing hidden Markov models to format the name and address data for record linkage. In addition, AddressDoctor could only deal with address inconsistencies using a large dynamic address library. There is a lack of available methods for domain agnostic data (non-specific domain in an uncertain context) representation consistency problems.

Current ETL (Extraction Transformation Loading) tools and data cleaning frameworks could deal with this domain agnostic problem. Domain agnostic method means these framework and tool could be used for data whatever the domain is, and, it still require some knowledge to distinguish the data features of different domains. The process of these approaches includes data analysis, definition of the transformation workflow and mapping rules, verification, transformation and backflow of cleaned data [42], which targets data quality issues (completeness, accuracy, consistency). However, it seems to use a big tool to solve a small problem for simple instance level data representation inconsistency problems. Users would need to read handbooks hundreds of pages long to learn to use ETL tools. In addition, this process requires a large amount information from users including metadata, instance level data characteristics, transformation mappings and workflow definitions.

This highlights a gap in both research and tools available for domain agnostic data representation inconsistency problems. This research is by no means trivial since the goal is to provide a declarative method to address domain agnostic data with less domain knowledge provided. Specifically, compared with current frameworks and ETL tools for domain agnostic data which requires large amount domain knowledge and complex programs, our approach requires only a set of seed patterns of the domain data. The set of seed patterns is easy to obtain through a quick scanning of the dataset by users. Followed by our framework, a complete and nearly mutually exclusive pattern library will be built up. This could be used for data from any domain.

Furthermore, we employ a declarative SQL extension instead of writing a massive amount of code. As an example, we look at the inconsistent stop data in Table 1.4. To attempt to repair this data to a consistent data format pattern, currently three kinds of methods are mostly used: 1) write several SQL "UPDATE" statements to change the string values one by one; 2) develop a integrity SQL function to find out the various stop format patterns and replace the substrings; or 3) employ other programming languages such as Java to alter the stop format pattern. These methods are inconvenient for most database users and require a certain level of expertise. Furthermore, in order to employ these methods, the format of these tuples must be known beforehand. For example, to write these SQL scripts, we would need to know the exact stop format pattern for each inconsistent tuple. Therefore, it is necessary to develop a declarative method (SQL extensions) to deal with this problem.

TABLE 1.4: Example: representations for “BoardingStops”.

BoardingStop	**Pattern***	**Consistent Data**
Wynnum Plaza - Stop 58 [BT006135]	*D* - Stop *SN* [*ID*]	Wynnum Plaza - Stop 58 [BT006135]
A.& I.I.C.S. - 55/56 [BT005196]	*D* - *SN* [*ID*]	A.& I.I.C.S. - Stop 55/56 [BT005196]
Alison St - St 32 [BT002904]	*D* - St *SN* [*ID*]	Alison St - Stop 32 [BT002904]
Trouts/Redwood - 40 [BT002172]	*D* - *SN* [*ID*]	Trouts/Redwood - Stop 40 [BT002172]
	SN - *D* [*ID*]	
Griffith University Stop A [BT010434]	*D* Stop *SN* [*ID*]	Griffith University - Stop A [BT010434]
	D *SN* [*ID*]	

* *D*, *SN*, and *ID* are fields where *D* represents stop description with regular expression $(\ |[A - Za - z0 - 9/\&.])^+$, *SN* represents stop number with regular expression $[A - Za - z0 - 9/]^+$, and *ID* represents stop ID with regular expression $BT[0 - 9]^+$. { ,-,Stop,St,[,]} are separators between fields.

1.3 Aims and Objectives

The objective of this thesis is to use SQL extensions and a pattern based framework to detect and repair instance level data representation inconsistency problems for domain agnostic data. Since our framework consists of three modules, improving data representational consistency can be divided into three subtasks, each of which constitutes an aim of this thesis:

1. Pattern design in SQL extensions: Pattern design construct a pattern library for each domain based on the iterative and interactive approach. We define data format or structure as a pattern, namely a sequence of fields and separators represented using regular expressions, as illustrated in Table 1.4 using SQL extensions shown in Chapter 4.

2. Pattern detection: Pattern detection is about finding inconsistent data representations in datasets. It belongs to the data profiling part of data cleaning tools, which detects and highlights data quality problems. For example, in Table 1.4, inconsistent boarding stop data can be detected using the pattern library for stop domain.

3. Pattern unification: After the inconsistent data and the patterns for each record is detected, automatic repair of this data is required. Pattern unification aims to transform data in various patterns into a target uniform representation in order to improve the data consistency of the

dataset. For instance, in Table 1.4, the inconsistent data needs to be changed to a consistent representation.

1.4 Challenges and Contributions

In this thesis, challenges abound in order to handle an inconsistent dataset both accurately and efficiently. First, the coverage and quality of patterns are critical. An incomplete pattern library will miss data records, while a badly-designed pattern library might cause conflicts between patterns. Therefore, an ideal pattern library should be complete and mutual exclusive. However, it requires extensive human efforts to construct such a pattern library from scratch. Second, pattern conflict means a data record can match multiple patterns. We observe two types of pattern conflict, namely field-field conflict where a substring maps to several fields (In Table 1.4, "Trouts/Redwood" in "Trouts/Redwood - 40 [BT002172]" can be a stop description as well as a stop number based on the regular expressions of *D* and *SN*), and field-separator conflict where a field covers a separator (In Table 1.4, we can regard "Griffith University" as a stop description and "Stop" as a separator in "Griffith University Stop A [BT010434]", but it is also possible to treat "Griffith University Stop" as a stop description). Hence, it is necessary to recognise such one-to-many mappings when conducting pattern detection. A straightforward approach is to adopt pairwise checking between data records and patterns which, however, is obviously very time consuming. Third, pattern unification is more complicated than string-based functions such as substring replacement. Consider "Alison St - St 32 [BT002904]" in Table 1.4 as an example. We cannot unify it to "Alison St - Stop 32 [BT002904]" simply by replacing "St" with "Stop". Instead, we need the semantic knowledge that "Alison St" as a whole denotes a stop description while the second "St" is a separator, and our goal is to unify only the separator "St" as "Stop". We tackle these challenges in this thesis, and implement a toolkit to improve data representational consistency with SQL extensions. More specifically, our contributions are summarized as follows:

- We design a SQL extension for improving data inconsistency, instead of writing massive SQL statements or complicated code specific to different domains.
- We propose a pattern based framework with a complete and nearly mutual exclusive pattern

library which is achieved by using an iterative and interactive approach.

- We construct a Finite State Machine (FSM) to recognise all possible patterns for each data record and meanwhile avoid pairwise checking.
- We introduce a two-level pattern definition to combine both domain knowledge and regular expressions, and propose a spilt-transform-merge method to facilitate pattern unification.
- We conduct a comprehensive evaluation on real-life datasets to verify the effectiveness and efficiency of our proposals, and implement a toolkit to improve instance level data representation inconsistency for domain agnostic data.

1.5 Thesis Organisation

The rest of this thesis is organized as follows. Chapter 2 provides the background and an overview of work related to data representation consistency and algorithms for string pattern matching. The research methodology for resolving or improving the data representation inconsistency problems is introduced in Chapter 3. In this chapter, we propose an user-driven pattern-based framework with the iterative and interactive approach and string pattern matching technology. The detailed methods about pattern design and pattern unification are introduced here. Chapter 4 details the design of the SQL Extensions and the pattern based method developed to improve data representational consistency with an use case. A modified DFA method for data inconsistency detection and for multiple string pattern matching is then presented in Chapter 5. This chapter also reports the evaluation result from the aspect of accuracy and efficiency of the pattern detection method. This is followed by a description of the implementation of the toolkit with an use case based on the methods above in Chapter 6. Finally, Chapter 7 concludes the thesis and outlines some directions for future research.

Chapter 2

Literature Review

This review aims to firstly summarize (a) the existing work related to data consistency; and (b) the algorithms for finding patterns in strings, and secondly, to find the links between the findings and my thesis. Also, data consistency is elaborated on from three different perspectives: concepts; framework and tools; and methodology. Finally, typical algorithms for three problems, (a) Single Keyword Algorithms, (b) Multiple Pattern Matching, and (c) Regular Expressions, will be expounded.

2.1 Data Inconsistency

Data quality issues have been widely researched from organizational, architectural, and computational aspects [44]. Data inconsistency is a significant problem in data quality, especially for enterprises. According to Lee et al. [33], many companies face a multitude of inconsistencies in data definitions, data formats and data values, which leads to difficulties in understanding and using data. Recently, many efforts have been made to improve data consistency. Specifically, in academia, various concepts, frameworks, and methods for data consistency have been discussed.

2.1.1 Background

As one of the fundamental dimensions of data quality [51, 52], data consistency has been studied from many perspectives. Roger and Paul [13] summarize three re-occurring aspects that appear throughout these perspectives, namely, the aspects of data integrity, semantics, and representational consistency. From these views of the dimension and data quality metrics, tools and computational methods have been developed which could solve the data inconsistency problems.

Data Consistency

Data consistency, as a significant dimension of data quality, can result in a myriad of problems due to the multiple aspects of it. These different aspects could be illustrated through definitions of data consistency. Ballou et al. [7] defined data consistency as the state where the "representation of the data value is the same in all cases". In 2003, they further defined it as format and definitional uniformity within and across all comparable data sets [8]. Gomes et al. [28] refer to Ballou and Pazers definition, which particularly included that the representation of that data is in a standard format. They defined inconsistent data as data that "doesn't convey heterogeneity, neither in contents nor in form".

These definitions come from a wide range of perspectives, and data consistency has become a complex dimension of data quality. In Wang's framework [52], data consistency was taken as representation, and was clustered with other dimensions as representational data quality. In 2007, Stvilia et al. [47] split consistency into intrinsic and extrinsic in their framework. Subsequently, they divided both intrinsic and extrinsic consistency into semantic consistency (same values for the same concepts and meanings) and structural consistency (same structure, format, and precision for similar values). Roger and Paul [13] summarize these perspectives into three aspects of data consistency: data integrity, semantics, and representational consistency.

Problem Classification

After a thorough survey of definitions and different aspects of data consistency in the literature, it is obvious that data inconsistency is not a simple problem. One way to classify the problem is according to the definition and dimensions of data consistency. That is to say, data inconsistency can be classified according to (a) data integrity inconsistency, (b) data semantic inconsistency, and (c) data representational inconsistency, for both structured data and unstructured data [13]. Also, such data classification can be based on when the inconsistency is generated. For example, data inconsistency that occurs in the midst of information integration could be classified into intentional and extensional [37]. From another angle, other studies [40, 42] provide the classifications and

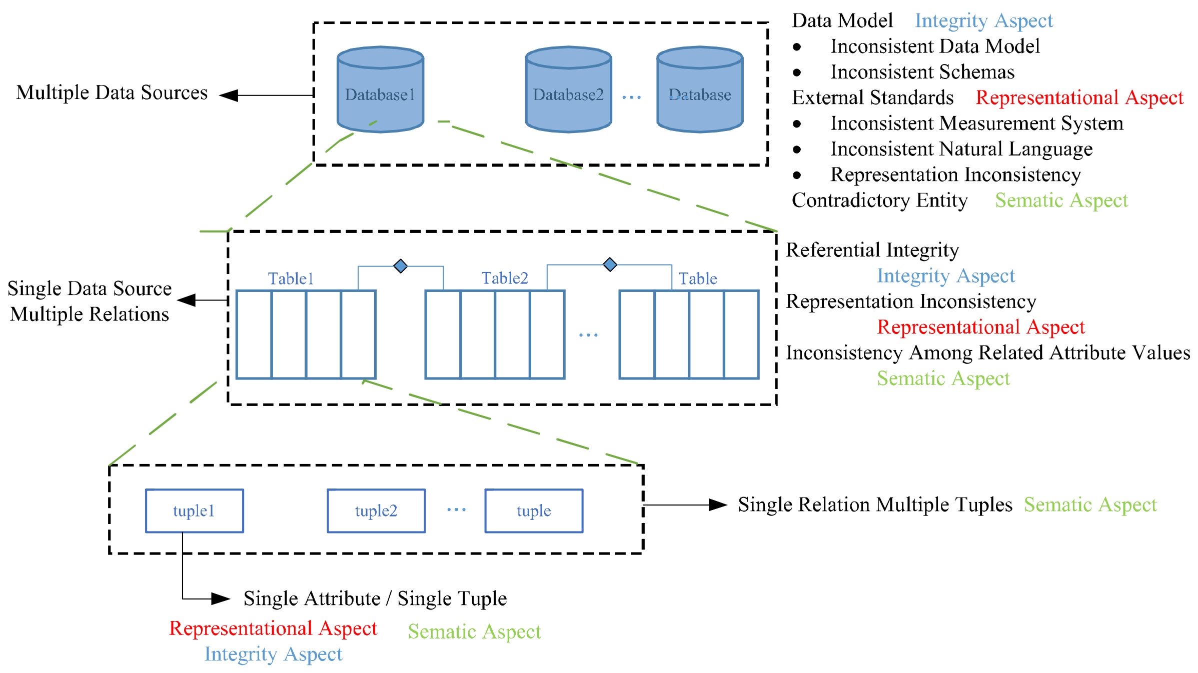
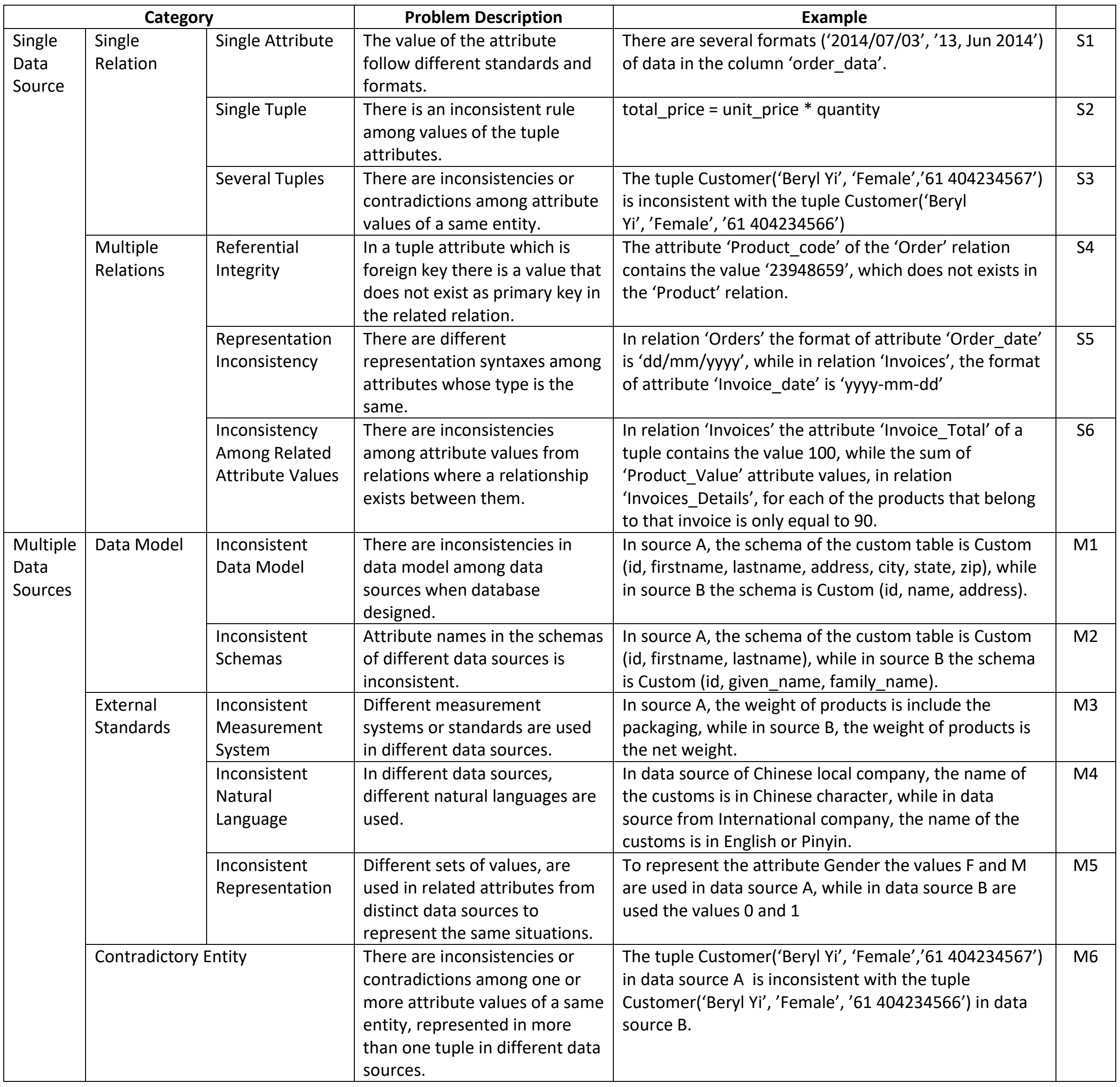

Data Model Integrity Aspect
• Inconsistent Data Model
• Inconsistent Schemas
External Standards Representational Aspect
• Inconsistent Measurement System
• Inconsistent Natural Language
• Representation Inconsistency
Contradictory Entity Sematic Aspect
Multiple Data Sources
Database1
Database2
Database
Single Data Source
Multiple Relations
Table1
Table2
Table
Referential Integrity
Integrity Aspect
Representation Inconsistency
Representational Aspect
Inconsistency Among Related Attribute Values
Sematic Aspect
tuple1
tuple2
tuple
Single Relation Multiple Tuples Sematic Aspect
Single Attribute / Single Tuple
Representational Aspect
Sematic Aspect
Integrity Aspect

FIGURE 2.1: Data consistency in relational database.

concepts of data quality problems including data inconsistency. In their study, Erhard and Hong [42] classify the major data quality problems to be solved by data cleaning and data transformation. With regards to the data cleaning process, they specifically distinguished between (a) problems between a single source and multiple sources, and (b) problems related to schema and instance. On the other hand, Oliveira et al. [40] identify and organize data quality problems via a bottom-up approach. In their paper, Oliveira et al. [40] identify the problem from the aspect of a single relation and multiple relations which we identify at an instance level and a schema level, respectively, for data representation inconsistency problems. In a relational model, data inconsistency problems are shown in Figure 2.1 below. From all these aspects of data inconsistency for different data quality problems, we summarize a classification for data inconsistency problems in a relational model in Figure 2.2. From all these aspects of data inconsistencies for different data quality problems, we summarize a classification for data inconsistent problems in relational model in Figure 2.1.



Category			Problem Description	Example	
Single Data Source	Single Relation	Single Attribute	The value of the attribute follow different standards and formats.	There are several formats ('2014/07/03', '13, Jun 2014') of data in the column 'order_data'.	S1
		Single Tuple	There is an inconsistent rule among values of the tuple attributes.	total_price = unit_price * quantity	S2
		Several Tuples	There are inconsistencies or contradictions among attribute values of a same entity.	The tuple Customer('Beryl Yi', 'Female','61 404234567') is inconsistent with the tuple Customer('Beryl Yi', 'Female', '61 404234566')	S3
	Multiple Relations	Referential Integrity	In a tuple attribute which is foreign key there is a value that does not exist as primary key in the related relation.	The attribute 'Product_code' of the 'Order' relation contains the value '23948659', which does not exists in the 'Product' relation.	S4
		Representation Inconsistency	There are different representation syntaxes among attributes whose type is the same.	In relation 'Orders' the format of attribute 'Order_date' is 'dd/mm/yyyy', while in relation 'Invoices', the format of attribute 'Invoice_date' is 'yyyy-mm-dd'	S5
		Inconsistency Among Related Attribute Values	There are inconsistencies among attribute values from relations where a relationship exists between them.	In relation 'Invoices' the attribute 'Invoice_Total' of a tuple contains the value 100, while the sum of 'Product_Value' attribute values, in relation 'Invoices_Details', for each of the products that belong to that invoice is only equal to 90.	S6
Multiple Data Sources	Data Model	Inconsistent Data Model	There are inconsistencies in data model among data sources when database designed.	In source A, the schema of the custom table is Custom (id, firstname, lastname, address, city, state, zip), while in source B the schema is Custom (id, name, address).	M1
		Inconsistent Schemas	Attribute names in the schemas of different data sources is inconsistent.	In source A, the schema of the custom table is Custom (id, firstname, lastname), while in source B the schema is Custom (id, given_name, family_name).	M2
	External Standards	Inconsistent Measurement System	Different measurement systems or standards are used in different data sources.	In source A, the weight of products is include the packaging, while in source B, the weight of products is the net weight.	M3
		Inconsistent Natural Language	In different data sources, different natural languages are used.	In data source of Chinese local company, the name of the customs is in Chinese character, while in data source from International company, the name of the customs is in English or Pinyin.	M4
		Inconsistent Representation	Different sets of values, are used in related attributes from distinct data sources to represent the same situations.	To represent the attribute Gender the values F and M are used in data source A, while in data source B are used the values 0 and 1	M5
	Contradictory Entity		There are inconsistencies or contradictions among one or more attribute values of a same entity, represented in more than one tuple in different data sources.	The tuple Customer('Beryl Yi', 'Female','61 404234567') in data source A is inconsistent with the tuple Customer('Beryl Yi', 'Female', '61 404234566') in data source B.	M6

FIGURE 2.2: Data inconsistency problems.

2.1.2 Related Frameworks and Commercial Tools

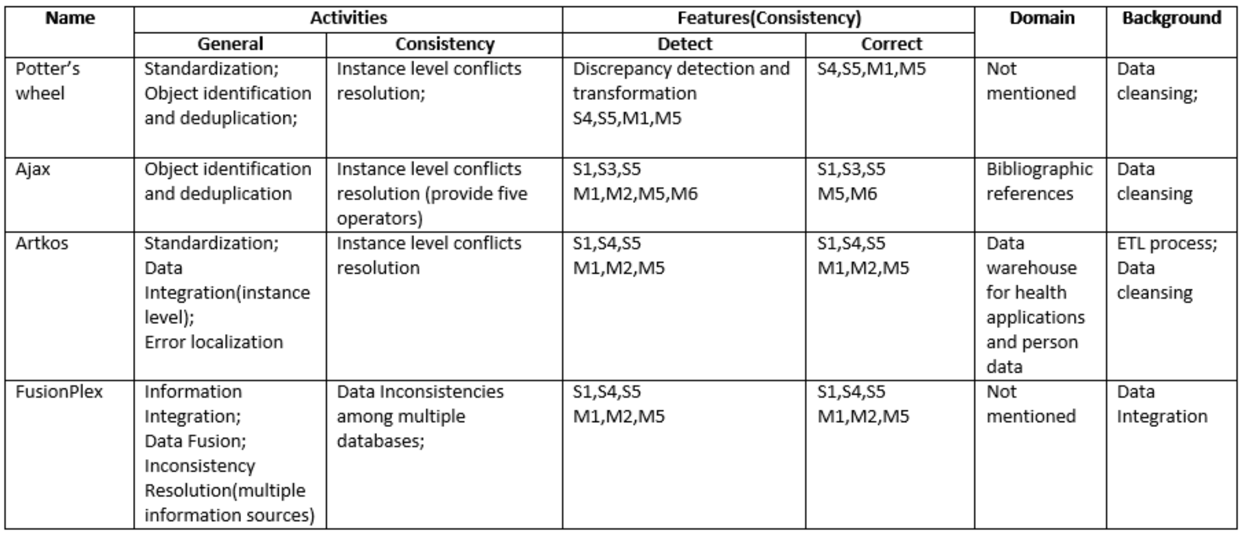
It is not difficult to conclude, from the previous section, that data inconsistency problems under different situations have been looked into from various aspects. Data inconsistency, as a significant data quality issue, has been mentioned in a great number of data cleansing and data integration frameworks. In addition, most ETL tools and BI tools include functions to deal with data inconsistencies. In general, these data quality frameworks and tools are either general purpose or special purpose. This section will discuss these frameworks and tools with a focus on data consistency.

General Purpose Frameworks

Data cleansing frameworks deal with data quality problems in general, rather than a tailored framework specifically for the problem of data inconsistency, thus treating data inconsistency as merely one of many data quality problems. Specifically, Potters Wheel [43] adopts a small set of transformation processes, such as “Format”, “Split” and “Merge” to analyse and clean the dirty data (e.g. schema or formats inconsistency and adherence to constraints in multiple data sources), with interactive methods applied. Although this method is not used to specifically address data inconsistency problems, it could deal with the problems (S5, M1, M2) listed in Figure 2.2. However, users have to program it using the complex transformation interfaces in Potters Wheel.

In terms of data integration, data inconsistency is a significant problem. As far as we know, many of the research studies that looked into issues of data inconsistency during data integration, focused mainly on data inconsistency among diverse information sources, especially, multi-database sources [36, 37]. For instance, FusionPlex [37], a system for integrating multiple heterogeneous information sources, has enhanced and extended the resolution of data inconsistencies of heterogeneous information sources with simple SQL extensions. This system includes two processes, namely, inconsistency detection and inconsistency resolution, using utility functions. It mainly focus on the problems resulting from multiple data sources (M1, M2, M5) as shown in Figure 2.2.

The problems (S3, M6) stated in Figure 2.2 have been researched in association with record linkage and duplication detection. Telcordias tool [15] places an emphasis on the importance of duplicate-record detection in performing record linkage. This tool is parametric according to distance and uses customized matching functions. However, it is not without its drawbacks as it focuses only on the error detection and data quality analysis, without editing inconsistent data. From the list of frameworks in Figure 2.3, Telcordias tool and Ajax have been engineered into commercial products. The rest of the frameworks remain as academic prototypes.



Name	Activities		Features(Consistency)		Domain	Background
	General	Consistency	Detect	Correct		
Potter's wheel	Standardization; Object identification and deduplication;	Instance level conflicts resolution;	Discrepancy detection and transformation S4,S5,M1,M5	S4,S5,M1,M5	Not mentioned	Data cleansing;
Ajax	Object identification and deduplication	Instance level conflicts resolution (provide five operators)	S1,S3,S5 M1,M2,M5,M6	S1,S3,S5 M5,M6	Bibliographic references	Data cleansing
Artkos	Standardization; Data Integration(instance level); Error localization	Instance level conflicts resolution	S1,S4,S5 M1,M2,M5	S1,S4,S5 M1,M2,M5	Data warehouse for health applications and person data	ETL process; Data cleansing
FusionPlex	Information Integration; Data Fusion; Inconsistency Resolution(multiple information sources)	Data Inconsistencies among multiple databases;	S1,S4,S5 M1,M2,M5	S1,S4,S5 M1,M2,M5	Not mentioned	Data Integration

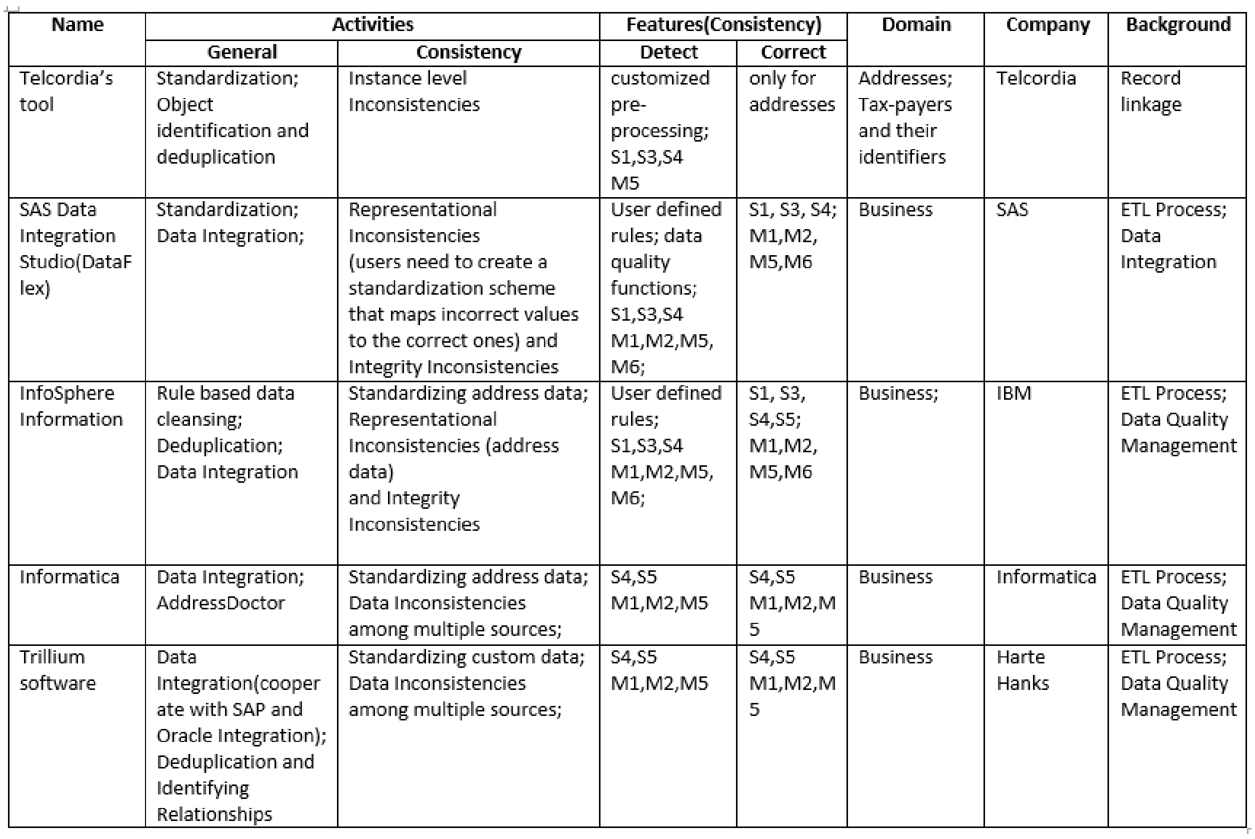
FIGURE 2.3: Frameworks related with data inconsistency.

General Purpose Commercial Tools

Organizations and companies today require a high level of data quality, so that their data analysis applications, e.g. decision support systems and customer relationship management (CRM) systems, can run efficiently. Meanwhile, enterprise cooperation and internationalization require a quality data integration process. There are plenty of commercial products [1] designed to address data quality problems, especially for data warehousing. These products provide data cleansing and data integration services including addressing business data inconsistencies. These products also support the transformation of data to be loaded into a data warehouse (ETL processing) via some data cleansing library functions. Barateiro and Galhardas [9] provide a survey of data quality tools used in both commercial and academic research fields. Having said that, we concentrate on the functionality and products related to data inconsistency. In general, these commercial products are applied: (a) for inconsistency detection and (b) for resolutions that address part of the data inconsistency problems. When they come across issues of data representation inconsistency, these

[1]IBM InfoSphere Information Server: http://www-01.ibm.com/support/docview.wss?uid=swg27008803; SAS(R) Data Integration Studio http://support.sas.com/documentation/cdl/en/etlug/66819/HTML/default/viewer.htm\#etlugwhatsnew4.htm; Informatica Data Integration http://www.informatica.com/us/products/data-integration/#fbid=LPthCBGKAaN; Trillium Software System http://www.trilliumsoftware.com/home/products/data-quality.aspx

tools are capable of standardizing address and customer names. To achieve the aim of a consistent dataset, users have no choice but to specify information for new domains in great details. Some famous commercial tools are summarized in Figure 2.4.



Name	Activities		Features(Consistency)		Domain	Company	Background
	General	Consistency	Detect	Correct			
Telcordia's tool	Standardization; Object identification and deduplication	Instance level Inconsistencies	customized pre-processing; S1,S3,S4 M5	only for addresses	Addresses; Tax-payers and their identifiers	Telcordia	Record linkage
SAS Data Integration Studio(DataFlex)	Standardization; Data Integration;	Representational Inconsistencies (users need to create a standardization scheme that maps incorrect values to the correct ones) and Integrity Inconsistencies	User defined rules; data quality functions; S1,S3,S4 M1,M2,M5, M6;	S1, S3, S4; M1,M2, M5,M6	Business	SAS	ETL Process; Data Integration
InfoSphere Information	Rule based data cleansing; Deduplication; Data Integration	Standardizing address data; Representational Inconsistencies (address data) and Integrity Inconsistencies	User defined rules; S1,S3,S4 M1,M2,M5, M6;	S1, S3, S4,S5; M1,M2, M5,M6	Business;	IBM	ETL Process; Data Quality Management
Informatica	Data Integration; AddressDoctor	Standardizing address data; Data Inconsistencies among multiple sources;	S4,S5 M1,M2,M5	S4,S5 M1,M2,M5	Business	Informatica	ETL Process; Data Quality Management
Trillium software	Data Integration(cooperate with SAP and Oracle Integration); Deduplication and Identifying Relationships	Standardizing custom data; Data Inconsistencies among multiple sources;	S4,S5 M1,M2,M5	S4,S5 M1,M2,M5	Business	Harte Hanks	ETL Process; Data Quality Management

FIGURE 2.4: Tools related with data inconsistency.

Special Purpose Tools

In addition to the above-mentioned general purpose frameworks and tools, there are a multitude of tools and methods available for standardization, which apply to specific data domains, e.g. addresses and personal name matching. These tools exist because data standardization is deemed as a prerequisite to achieving semantic consistency. Data standardization is necessary in order to address data representation inconsistency problems (S1, S5, M5). Standardization tools deal with these problems in basic and relatively straightforward ways. They usually rely on two components: (a) a set of data format rules, and (b) a transformation function library. The differences among these vendor tools include the level of complexity of the rules and the transformations allowed. In terms of methodology, knowledge based methods and methods customized for specific domains are employed in the tools. Standardization tools (e.g. AddressDoctor [2]) are designed to support more functions in the specific data domain (e.g. addresses). These tools are equipped with (a) domain-specific knowledge base methods and (b) string matching methods to address inconsistencies.

There are huge numbers of tools and frameworks available to address data consistency from different perspectives. We shortlisted some typical frameworks and popular commercial tools and summarized the data inconsistency problems addressed by these tools in Figure 2.3 and Figure 2.4. According to these figures, these commercial tools and frameworks mainly focused on: (a) data standardization, (b) data deduplication, and (c) data integration. Standardization deals with partial representation inconsistencies (S1, S5, M5) of address and custom data. Deduplication and object identification address semantic inconsistency problems (S3, M6).

2.1.3 Methodology for Different Data Inconsistency Problems

In order to improve the data quality, the traditionally methodology is through procedural solutions especially for specific problems and domains. That means they analyse the data to identify the root causes of data quality problems. Strategies that are driven by both data and process were introduced by Batini and Scannapieco [11] as methodologies to classify data quality. In recent years, there has been more research into generic solutions, and declarative and rule based specifications of data cleaning processes. Taking into consideration the large quantity of literature and methods related to data consistency (e.g. record linkage and data integration), in this section we review the approaches that address different aspects of data inconsistency problems.

Classic Data Consistency and Integrity Constraints

Issues of data integrity inconsistency in relational databases have been researched for many years. An integrity inconsistency means that within the database there exists data for which data integrity constraints cannot be satisfied. The problems could be due to several reasons related to integrity constraints in a DBMS [44]. For example, poorly designed or implemented applications which fail to maintain the consistency of the database, or integrity constraints that are enforced for better

[2]https://www.informatica.com/addressdoctor.html#fbid=6yEsoGD6SYX

performance of application programs or DBMSs, or integrity constraints that are just assumed to be satisfied based on knowledge about the application domain and the kind of updates performed on the database. In the area of data integration, data integrity consistency is much more difficult to achieve [12, 34]. The source data could be from different autonomous databases that are separately consistent with respect to their own local integrity constraints but when the data is integrated, new data integrity consistency issues may arise.

Integrity constraints have been studied in general and have wide applications in data management There are a few ways to achieve data consistency. One of them consists of declaring the integrity constraints together with the schema, and the DBMS will take care of the database maintenance, which is done by rejecting transactions that violate the constrains. This method could be automatically supported by most commercial DBMSs; however, the classes of integrity constraints supported are usually quite restricted [49].

Another way to maintain data consistency is based on the use of triggers that are stored in the database [16]. The reaction to a potential violation is programmed as the action of a trigger. Consistency could also be enforced through the application programs that interact with the DBMS. However, in no way can a DBMS guarantee the correctness of triggers or application programs and ensure database consistency. There has also been recent work done on constraint repair, which specifies the consistency of data in terms of constraints, and detects inconsistencies in the data as violations. These are mostly based on traditional dependencies, which are developed mainly for schema design.

Methods for Semantic Inconsistency Problems

We have considered the classic integrity constraints (functional dependencies and referential constraints) as methods to ensure data consistency. However, these constraints are not always expressive enough to represent the relationships among values for different attributes in a table, and are insufficient to capture the semantics of the data. Therefore, conditional functional dependency (CFD) is introduced to capture and repair data inconsistencies in relational databases. Fan et al. [22] proposed the conditional functional dependency and looked into its applications in data inconsistency. They argued that CFDs aim to capture the consistency of data by enforcing bindings of semantically related values, based on their previous work on CFDs. Later, Chen and Fan [17] extend the CFDs by considering cardinality and synonym rules. Since then, CFDs have been further extended to consider ranges of values, and pattern tables have also been employed to show the portions of the data that satisfy a constraint [27]. These papers are mainly focused on the detection of semantic inconsistencies using CFDs, without providing methods to fix the inconsistencies. In 2010, editing rules [23] based on conditional functional dependencies, were introduced for the process of data monitoring to repair data and guarantee that the repairs are correct. However, editing rules require users to have an in-depth knowledge of the data including feature details and also schema details to examine tuples to verify repairs and, importantly, to ensure new errors are not introduced, which can be costly. Fixing rules are introduced to prevent users from triggering repair operations by activating both evidence patterns and negative patterns [50].

Methods for Data Representational Inconsistency

Data representation consistency refers to data of the same values having uniformity of format [52]. The methods for solving data representation inconsistencies have been discussed in different data quality contexts such as data cleaning frameworks [43] and data fusion [[37]. These methods specifically pay attention to schema inconsistencies where schema conflicts exist across datasets or different information sources, and in particular, different schema representations of the same object in different tables. This method demands extensive prerequisite knowledge of the schema from the target datasets. Another set of methods is applied to address issues of data representation inconsistency for certain domain data within a specific context. For example, the healthcare sector can be a specific domain, where data is required to be exchanged among professionals and institutions. Churches et al. [18] propose using hidden Markov models to format name and address data for record linkage. Lexicon-based tokenization is used to split the strings. Hidden Markov models are trained to standardize typical Australian name and address data drawn from a range of health data collections. In contrast to methods applied to specific domain data, there is gap in available methods to solve data representation inconsistencies in domain agnostic data.

In this section we introduce the methodologies from a computational perspective, which can be further divided into three types of functionality. The first of these monitors and restricts inconsistent data by using classic integrity constraints. This method has been well researched in the area of data integrity in the relational model. Another popular methodology applied for inconsistency detection and repair uses conditional functional dependency. This method looks at the data inconsistency issue from, specifically, a semantics aspect. From a data representation aspect, there are some methods available that focus on certain domains, e.g. addresses, or phone numbers. Nevertheless, methods for domain agnostic data are understudied.

2.2 Algorithms for Finding Patterns of Strings

2.2.1 Problem Background

The problem of finding patterns of strings can be considered as an extended problem of string pattern matching. The difference between the two types of problems lies in the likelihood of having multiple patterns that can match with one single string. String pattern matching is one of the key problems in many fields of science and information processing [4]. Although data is memorized in various ways, text is still the main form of data used to exchange information. This is particularly true on websites, where a large amount of information is stored as textual data. With a large amount of data stored in a sequenced text file, string matching problems happen not only in computer science but also arise when analysing molecular phylogeny and molecular biology. In addition, string matching algorithms are basic components used in the implementation of software especially in text searching modules. Various algorithms exist for string pattern matching problems, e.g. the KnuthMorrisPratt (KMP) algorithm [30] and the BoyerMoore (BM) algorithm [14], most of which are devoted to improving the efficiency of the matching process.

The basic string matching problem is to locate all the occurrences of a given pattern $P = \{p_1, p_2, ..., p_m\}$ (m represents the length of the pattern) in a text $T = \{t_1, t_2, ..., t_n\}$ (n represents the length of the text), where both T and P are sequences of characters from a finite character set Σ. Given strings x, y, z, we say that x is a prefix of xy, a suffix of yx, and a factor of yxz [39].
Many algorithms deals with the above-mentioned problem and many studies have been undertaken for faster and simpler algorithms since 1977 [39]. To describe those algorithms clearly, we classify the string pattern matching algorithms in line with the number of patterns used. In this section, we classify the algorithms into three groups: (a) Single Keyword Algorithms, (b) Multiple Pattern

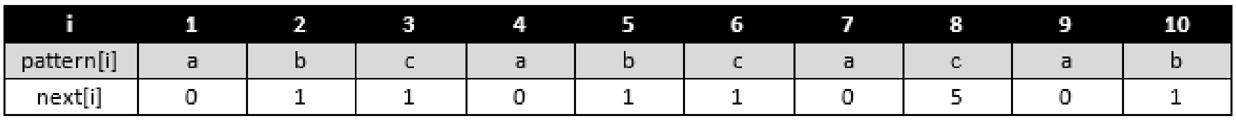
Matching Algorithms, which focus on a finite set of patterns, and (c) Regular Expressions, which look into an infinite number of patterns.

2.2.2 Single Keyword Algorithms

Single keyword matching means to locate all occurrences of a given pattern in the input text string [5]. The naive algorithm is called brute-force(BF), which is the simplest technique for single keyword matching. This algorithm scans the text from left to right and checks the characters of the keyword pattern character by character. However, the worst-case time required for determining that the pattern does not occur in the text is $O(mn)$ (m represents the length of the pattern, and n represents the length of the text) which is not very efficient. In order to improve the efficiency over the BF technique, plenty of methods have been proposed. The oldest and most famous are the Knuth-Morris-Pratt(KMP) and the Boyer-Moore(BM).

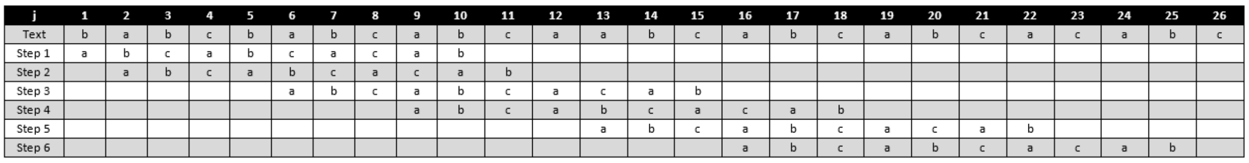
Knuth-Morris-Pratt(KMP) Algorithm

KMP algorithm (Knuth-Morris-Pratt), as the best known for linear time for exact string matching, was first proposed by Donald Knuth and Vaughan Patt and independently by James H.Morris in 1977 [30]. This algorithm is $O(n)$ in the worst and average case for the searching phase, and the preprocessing complexity is $O(m)$ [39]. The KMP algorithm introduce a shift table $next[]$ compared with the BF algorithm in order to shift more characters(shift $i+1-next[i+1]$ characters, i represents the position in the pattern.) . Consider the following example in Figure 2.6. The keyword pattern is "$abcabcacab$" and the shift table $next[]$ shown in Figure 2.5. As shown in Figure 2.6, after 5 shifts, the pattern "$abcabcacab$" is matched to the target text. Taking step 3 as an example, at that point in time the current position for the text is $j = 6$ where the pattern starts matching from the fist character. When $j = 13$ and $i = 7$, we find current positions for the text is a, while for the pattern it is 'c', where matching becomes unsuccessful. We could see in Figure 2.6 that $next[8] = 5$. Hence, the pattern should shift $8 - next[8] = 3$ characters. In step 4 in Figure 2.6, the fifth character in pattern is matched with the text when $j = 13$. Through this example, it is not difficult to realize that the key to this algorithm is how to generate the next table. Many variants exist based on the KMP algorithm that try to optimize the next table [30]. The most important one



i	1	2	3	4	5	6	7	8	9	10
pattern[i]	a	b	c	a	b	c	a	c	a	b
next[i]	0	1	1	0	1	1	0	5	0	1

FIGURE 2.5: Next table.



j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Text	b	a	b	c	b	a	b	c	a	b	c	a	a	b	c	a	b	c	a	b	c	a	c	a	b	c
Step 1	a	b	c	a	b	c	a	c	a	b																
Step 2		a	b	c	a	b	c	a	c	a	b															
Step 3						a	b	c	a	b	c	a	c	a	b											
Step 4									a	b	c	a	b	c	a	c	a	b								
Step 5													a	b	c	a	b	c	a	c	a	b				
Step 6																a	b	c	a	b	c	a	c	a	b	

FIGURE 2.6: Example of KMP.

is the Simon algorithm [45], which shows that the automaton of KMP can be completed and stored in an efficient way.

Boyer-Moore(BM) Algorithm

The BM Algorithm is the fastest pattern matching algorithm for a single keyword in both theory and practice [14]. Using the KMP algorithm, the pattern is scanned from left to right, but the BM algorithm compares characters in the pattern from right to left. If mismatch occurs, then the algorithm computes the amount by which the pattern is moved to the right before a new matching.

We refer again to the example in Figure 2.6 to explain the difference between the KMP and BM algorithms. In step 3, if we use the KMP algorithm, we will move to the fifth character in the pattern matching with the text when $j = 13$. In this position in the text, the character is 'a'. If the character turns out to be 'd', the characters in the pattern cannot be matched successfully because there is no character 'd' in pattern "abcabcacab". Therefore, in the BM algorithm, when 'd' is not found in the pattern, the position of the text could move directly to $j = 14$, and then match with the first character in the pattern. This means a bigger shift when matching. That is why the BM algorithm produces faster matching. The main reason for this is that it includes a two shift table.

The worst case complexity of the matching process is $O(mn)$. As for the KMP algorithm, there are also many variants of the BM algorithm. Baeza-Yates formalized the concept of a BM automaton and presented an efficient algorithm in 1994 [6].

2.2.3 Multiple Pattern Matching

The single string matching problem may be extended in a natual way to search for a set of strings $P = \{p^1, p^2, ..., p^r\}$, where each p^i is a string $p^i = p^i_1 p^i_2 ... p^i_m$ over a finite character set Σ [39]. The sum of the lengths of the strings in P could be represented as $|P| = \sum_{i=1}^{r} m_i$. As above, the search is done in a text T. For example, if we search for the pattern set $\{ABABA, BABA\}$ in a text, each time we find an occurrence of $ABABA$ we also find an occurrence of the second string $BABA$. Therefore, the total number of occurrences can be $r * n$.

The simplest method is to repeat r searches with one of the algorithms of single keyword pattern matching. This leads to a total worst case complexity of $O(r * n)$ for the matching processing. Since multiple pattern matching is an extension of single keyword pattern matching, the algorithms adopted are also extensions of the KMP and BM algorithms. An extension of the KMP algorithm produced the AhoCorasick algorithm, while the Commentz-Walter algorithm is based on the BM algorithm.

Aho-Corasick string matching algorithm

The Aho-corasick(AC) algorithm employs a special automaton, called Aho-corasick automaton, which is built on P [2]. This algorithm is processed as that the text string is scanned from the left to right in a single pass. The automaton comprises a finite set of states together with the rules how it moves from state to state.

The classical AC algorithm includes three functions: goto function, failure function and output function. **The goto function** maps a state character pair into a state or message of fail. **The failure function** is a state to state mapping, which is consulted whenever the goto function reports a failure. **The output function** formalizes this concept by associating a subset of keywords with every state. We use an example devised by Aho and Corasick [2] to explain how this works (refer to Figure 2.7. In this example, the pattern set is $P = \{he, she, his, hers\}$. The suffix of pattern she is the pattern he, while the pattern he is also the prefix of pattern $hers$. Therefore, if pattern she could be matched in the position of $j, j+1, j+2$, it means the position of $j+1, j+2$ could match pattern he and the first two characters of pattern $hers$. Therefore, we just need to align at the third character of pattern he and $hers$ without back tracking to the current position j. Theoretically, the automaton may be constructed from the set of keyword strings in $O(m)$ time, and used to search the text string in $O(n)$ time.

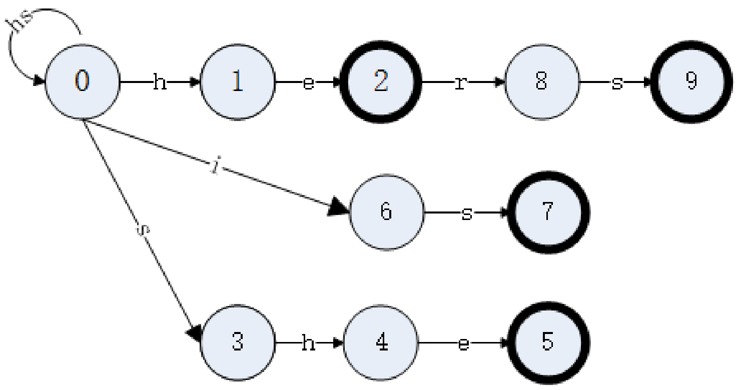


FIGURE 2.7: Example of AC.

Commentz-Walter algorithm

The Commentz-Walter algorithm, as the first expected sub linear multi-string matching algorithm, also uses the automaton technique and is a generalisation of the BM approach to string matching. Historically, it was implemented in the second version of the Unix application Grep. However, currently, there are no real cases of applications using this algorithm, and we just introduce the idea it is based on. In the Commentz-Walter algorithm, an automaton for the set of the reversed keywords is constructed [20]. Although this method has been shown to be faster in practice than the AC algorithm for small numbers of pattern strings, in the worst case it does run in quadratic time [46].

2.2.4 Regular Expression

After presenting an overview of finite string patterns, we will discuss the widely used infinite string pattern matching method. Regular expressions are often used to represent search patterns that are more complex than strings such as an infinite string set. The following section will introduce the basic regular expression approaches and the literature on multiple regular expression rules.

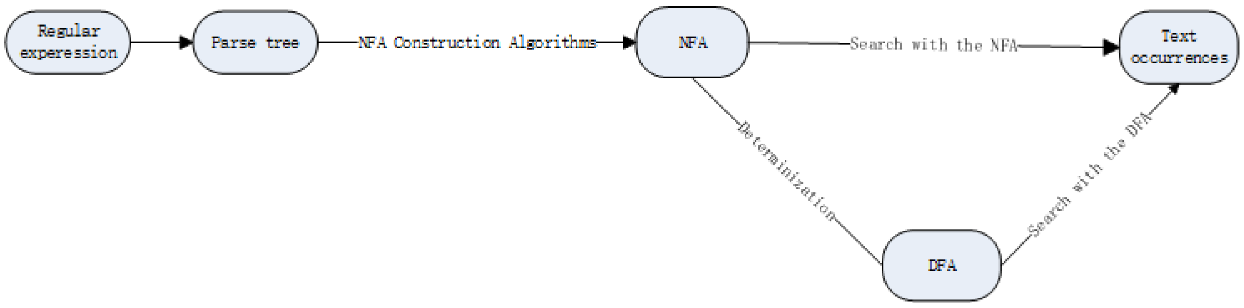


FIGURE 2.8: Regular expression matching in a text.

Basic Regular Expression Approaches

The classical approaches is summarized in Figure 2.8. The regular expression is first pared into an expression tree, then a nondeterministic finite automaton (NFA) is constructed. In order to improve the searching efficiency, some regular expression engines transform an NFA into a deterministic finite automaton (DFA). Finally, the NFA or DFA is used to perform pattern matching.

1. NFA Construction Algorithms: There are various algorithms to build an NFA from a regular expression [39]. It is not uncommon to find the Thompson construction [48] and the Glushkov construction [26] been regularly applied in practice. The Thompson construction [48] points to an NFA which is linear in its number of states (at most $2m$) and of transitions (at most $4m$). The Glushkov construction [26], on the other hand, points to an NFA with precisely $m + 1$ states but numerous transitions that is $O(m^2)$ in the worst case.

2. Regular Expression Searching Approaches: : As shown in Figure 2.8, there are two branches: NFA and DFA. On the basis of the direct simulation of his NFA, Thompson proposed the search algorithm NFAThompson [39]. In this algorithm, the set of active states are activated by the current text character and represented in a suitable way such as a bit vector, stored explicitly. The DFA technique comes with two parts: the regular expression being translated into a DFA and the DFA employed for text searching [39]. Compared with an NFA, when we traverse the text, a DFA has exactly one active state at a time. Instead of a pure NFA or DFA, a hybrid approach, which is an intermediate between a DFA and an NFA, is proposed [38]. This approach is based Thompsons construction and consists of splitting the NFA into modules, making each of them deterministic, and keeping it an NFA as a whole.

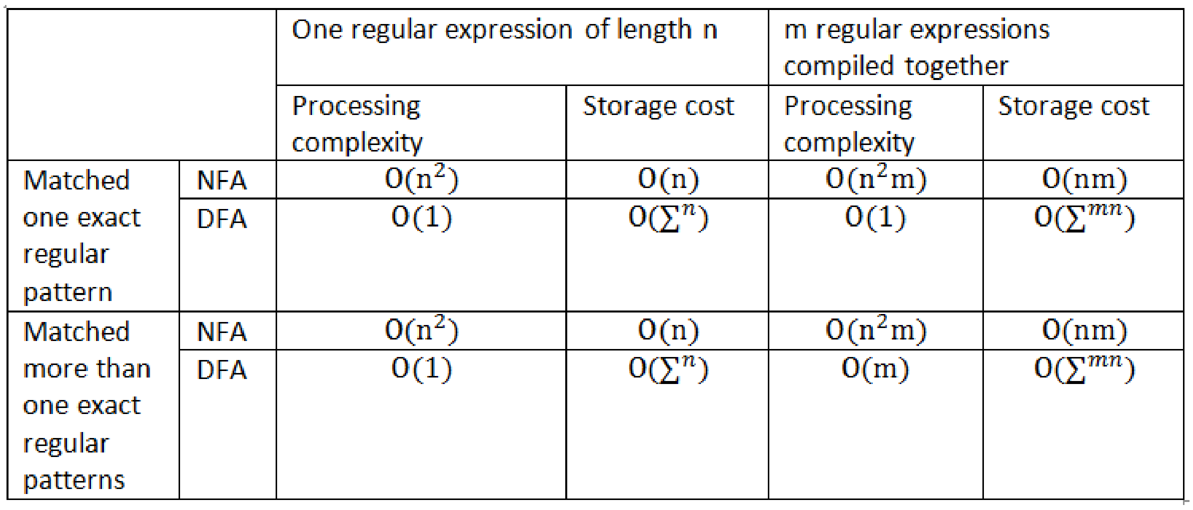
Multiple Regular Expression Rules

Multiple regular expression rule matching could be considered as an extension of multiple pattern matching for regular expression patterns. Instead of moderate sized pattern sets (e.g. in q-Grams based BoyerMooreHorspool algorithms [32] and Backward algorithms which combine the BM heuristic idea and the AC automation idea [19]), the literature indicates that multiple regular expression rule matching deals efficiently with larger pattern sets [55]. Multiple regular expression rules have been studied a lot in the context of URL filtering and Network Intrusion Detection (NID) systems. Specifically, an algorithm called TFD is proposed, which employs a two-phase hash, a FSM and double-array storage to exclude the performance bottleneck of backlist filters, to achieve large-scale and high-speed URL filtering [54]. Literature on NID has specifically focused on the optimization of the DFA, to speed up the matching and to reduce memory requirements. For example, Yu et al. [53] proposes a regular expression rule rewrite method and grouping solutions to improve multiple regular expression rules from both speed and memory usage aspects. Later in 2015, the Templates Finite Automata Grouping Algorithm (TFA) was introduced to segregate rule sets into different groups in order to reduce the number of rules [35].

Discussion of Solution Space using DFA and NFA

In summary, finite automata are usually used to represent regular expressions. There are two main categories of finite automata: Deterministic Finite Automaton (DFA) and Nondeterministic Finite Automaton (NFA). Here, we discuss the solution space using DFA and NFA for theoretical matching according to the analysis in the studies by Navarro and Raffinot [39] and Yu et al. [53].

To handle m regular expressions and find all the possible matched regular expressions, two options are possible: processing them individually in m automata, or compiling them into a single automaton. Recent works have proposed the latter approach, so that the single composite NFA can support shared matching of common prefixes of those expressions. Although it has demonstrated performance gains over using m separate NFAs, this approach generates a large number of active states. The resulting case complexity from this approach is as bad as for the sum of m separate NFAs. Therefore, this approach can be slow in our context, because given any input character, each active state must be serially examined to obtain new states. In a DFA-based engine, compiling m regular expressions into a composite DFA results in definite performance benefits over running m individual DFA. Specifically, when it comes to a situation where only one possible regular pattern is matched, a composite DFA reduces the processing cost from $O(m)$ to $O(1)$) (i.e. a single lookup to obtain the next state for any given character). However, the number of states in the composite automaton grows to $O(\Sigma^{mn})$ in the theoretical worst case. In our context, there may be at most m regular expressions found. Therefore, the worst case could still cost $O(m)$ for m possible regular patterns. Figure 2.9 shows the worst case comparisons of DFA and NFA approaches. In order to ensure the processing speed of regular expression matching, we choose a DFA to deal with the inconsistency detection problem.



		One regular expression of length n		m regular expressions compiled together	
		Processing complexity	Storage cost	Processing complexity	Storage cost
Matched one exact regular pattern	NFA	$O(n^2)$	$O(n)$	$O(n^2m)$	$O(nm)$
	DFA	$O(1)$	$O(\Sigma^n)$	$O(1)$	$O(\Sigma^{mn})$
Matched more than one exact regular patterns	NFA	$O(n^2)$	$O(n)$	$O(n^2m)$	$O(nm)$
	DFA	$O(1)$	$O(\Sigma^n)$	$O(m)$	$O(\Sigma^{mn})$

FIGURE 2.9: Worst case comparisons of DFA and NFA.

2.3 Summary

2.3.1 Data Inconsistency

Based on the data quality problems summarized in Figure 2.2 and the studies of frameworks and tools (refer to Figure 2.3 and Figure 2.4), we conclude that there is a lack of literature on representation inconsistency problems at an instance level for domain agnostic data. In contrast, academic studies on integrity data consistency have been well dealt with and findings have been applied to commercial DBMSs, e.g. IBM and SAP. The research on semantic data consistency has, so far, attracted most of the attention from researchers. One of the prominent methods used to resolve semantic data inconsistency problems is CFD; this is useful when looking at records that have a dependency relationship.

Currently, the data representation inconsistency issue is being looked into from both academic and industrial perspectives. Firstly, academic research on data representation inconsistency problems at an instance level, to certain degree, is confined within a certain domain in a specific context, which we refers to as domain specific. A typical example is the health-care sector, within which a great deal of patient heath data is exchanged among different parties, e.g. healthcare professionals and institutions. Churches et al. [18] proposed to specifically format the name and address data for record linkage by utilizing a hidden Markov model. In addition, tools like AddressDoctor cannot deal with the problem of address inconsistencies without access to a large dynamic address library.

Secondly, industrial players applied Extraction Transformation Loading (ETL) tools to solve the problem and data cleaning frameworks are able to deal with this domain agnostic problem. The approaches have processes that include (a) data analysis, (b) definition of the transformation workflow, (c) mapping rules, (d) verification, (e) transformation, and (f) backflow of cleaned data. Instead of solving only consistency, a number of other problems, such as accuracy and redundancy, can also be solved with the above-mentioned process in place. Nonetheless, it turned out to be a complicated tool for a small area of instance level data representation inconsistency problems. To use these complex ETL tools, users must read and learn from handbooks that are hundreds of pages long. On top of that, in order to use this process, a great deal of information is required from the user end, e.g. metadata, instance level data characteristics, transformation mappings and workflow definitions.

Thus, a gap has been identified in both academic research and industrial tools that address the domain agnostic data representation inconsistency problem. This research, therefore, is of great importance as it is dedicated to provide a declarative method to address domain agnostic data in a convenient way. That is to say, by implementing a smart tool, less input and expertise is required from the user, in contrast to the available ETL tools. Moreover, we employ a declarative SQL extension, rather than writing a large volume of code.

2.3.2 Algorithm for Finding Patterns of Strings

There are plenty of algorithms that can be applied to string pattern matching problems. We have introduced some typical algorithms and provided details about them. The main approaches for algorithm improvement fall into two categories: (a) skip more characters when matching and (b) reduce the traversal times. For tasks such as multiple pattern matching and regular expression matching, most algorithms scan the target string in a single pass with automaton technology. Particularly, with s to regular expression matching, efforts have been made to optimize algorithms by reducing the number of states. However, for strings which are matched with more than one pattern, the information about the patterns tends to be missed when algorithms attempt to reduce the number of states. Therefore, when the string was eventually matched, these algorithms emphasized the location where string matched, instead of identifying which patterns in the string were matched. Some other algorithms have paid attention to the issue of patterns through group states instead of optimizing the algorithms. In my thesis, we use a modified subset construction algorithm with a break function added on. This modified algorithm is meant to avoid the issues of having pattern information disappear during the optimizing of the DFA.

Chapter 3

Research Methodology

In this chapter, we introduce a user-driven pattern-based framework which is based on a)iterative and interactive approach, and b)string pattern matching technology. The iterative and interactive method aims to achieve a complete and nearly mutual exclusive pattern library. String pattern matching technology is used to generate pattern detection and data profiling results. In the following section, we will a)show the preliminaries and problem statement of this thesis, b)propose the pattern based framework, and c)describe the approaches used in the pattern based framework.

3.1 Preliminaries and Problem Statements

In order to recognise representational inconsistency in data, we propose a two-level pattern definition in this thesis to reflect the format or structure of data. Specifically, a pattern is defined in both semantic level and lexical level. At semantic level, a pattern can be regarded as a sequence of fields and separators. While at lexical level, a pattern is a sequence of regular expressions.

Definition 3.1 (Pattern). *A pattern p is represented as a sequence of fields and separators, namely $p = (f_1, s_1, f_2, s_2, f_3, ..., f_{t-1}, s_{t-1}, f_t)$ where each f_i and s_i denote a field and a separator respectively, both of which are expressed as regular expressions.*

We denote the set of fields and the set of separators as $\mathbb{F} = \{f_1, f_2, ..., f_m\}$ and $\mathbb{S} = \{s_1, s_2, ..., s_n\}$ respectively. Consider the example in Table 1.4. The field set is $\mathbb{F} = \{D, SN, ID\}$, and the separator set is $\mathbb{S}$ =$\{$␣,-,Stop,St,[,]$\}$. From the field set and the separator set, we can construct several patterns such as "*D* - Stop *SN* [*ID*]", "*D* - *SN* [*ID*]", "*D SN* [*ID*]", etc. We denote the pattern library as $\mathbb{P} = \{p_1, p_2, ..., p_k\}$.

For each data record d from a dataset $\mathbb{D}$, we denote the set of patterns it maps to as $\mathbb{P}_d = \{p_i | p_i \in \mathbb{P} \wedge d \rightarrow p_i\}$ where $d \rightarrow p_i$ means that data record d can match pattern p_i. Hence, the set of patterns that dataset $\mathbb{D}$ contains can be denoted as $\mathbb{P}_{\mathbb{D}} = \bigcup_{d \in \mathbb{D}} \mathbb{P}_d$.

Definition 3.2 (Data Representational Inconsistency). *A dataset $\mathbb{D}$ is representation inconsistent when it contains multiple patterns, namely $|\mathbb{P}_{\mathbb{D}}| > 1$ where $|\mathbb{P}_{\mathbb{D}}|$ represents the size of (or number of patterns contained in) $\mathbb{P}_{\mathbb{D}}$.*

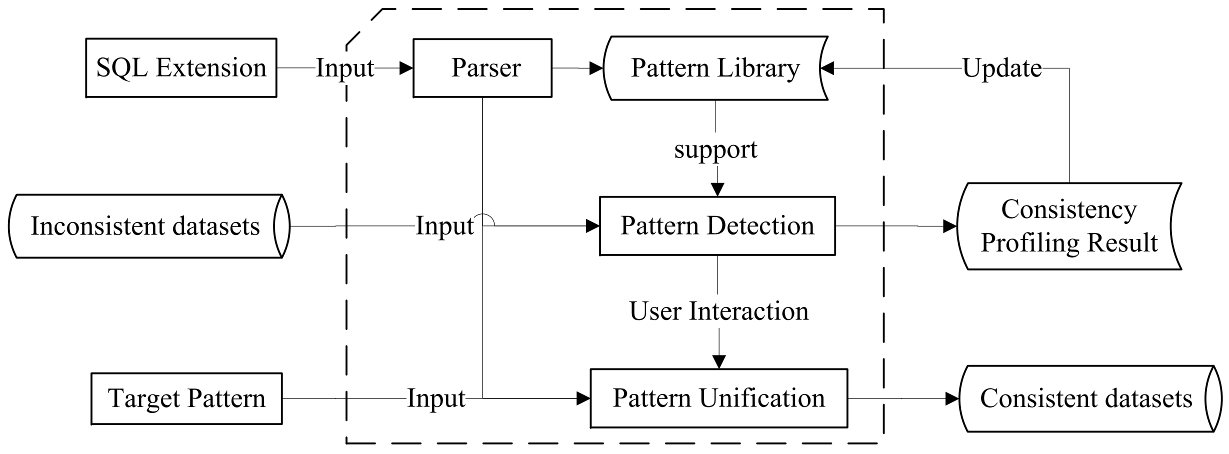


FIGURE 3.1: An overview of framework.

3.2 Pattern Based Framework

The goal of improving data representational inconsistency is to unify an inconsistent dataset to a single pattern. Figure 3.1 demonstrates the framework proposed in this work, which consists of three modules: pattern design, pattern detection, and pattern unification.

The pattern design module constructs a pattern library for each data domain. We adopt an iterative and interactive approach for pattern design. We will discuss the details of pattern design in Section 3.3. Given a dataset $\mathbb{D}$ and the pattern library $\mathbb{P}$, the pattern detection module recognises possible patterns $\mathbb{P}_d$ for each data record $d \in \mathbb{D}$. We will discuss technology of pattern detection in Section 3.4. The details and evaluations will be illustrated in Chapter 5. If the dataset $\mathbb{D}$ is representational inconsistent (i.e., $|\mathbb{P}_{\mathbb{D}}| > 1$), the pattern unification module will be triggered. It receives a target pattern p^* from the user, and transforms all data records into this pattern to make the dataset $\mathbb{D}$ consistent. We will discuss the details of pattern unification in Section 3.5. Note that when a data record d maps to multiple patterns, we need to pick a pattern p_i from the pattern set $\mathbb{P}_d$ and conduct unification based on the specific pattern p_i. This can be done randomly. However, if a semantically incorrect pattern is chosen as the unification pattern, it will cause the resulting data record to be semantically incorrect. Take “Trouts/Redwood - 40 [BT002172]” in Table 1.4 as an example. Assume “*D* - Stop *SN* [*ID*]” is the target pattern and we select “*SN* - *D* [*ID*]” as the pattern for this data record, then the unification result will be “40 - Stop Trouts/Redwood [BT002172]” which is obviously wrong. Hence, a better solution is to ask users to select the best pattern when multiple patterns are detected. But in order to reduce the amount of user interactions, it also requires the pattern library to be less conflicting.

3.3 Pattern Design

3.3.1 The virtuous cycle of iterative and interactive approach

As discussed in Chapter 1, most ETL tools require a great deal of information from the user, including metadata, instance level data characteristics, transformation mappings and workflow definitions. However, in practice, the user may not have sufficient knowledge about the target dataset. The virtuous cycle of iterative and interactive approach is designed for that situation. It means users need only provide the structure information of the possible string patterns, which are represented by regular expressions, without requiring knowledge of transformation mappings and workflow definitions. In addition, this cycle can guide users to build the pattern library complete and nearly mutually exclusive. After the data profiling module is processed, the results (conflict patterns and unmatched patterns) returned could guide the user to discover additional different patterns, complete the regular expressions, and assure the correctness of pattern matching results. This continuous user interaction and data profiling leads to an overall improvement in the accuracy of the profiling results, and avoids introducing errors in the unification module.

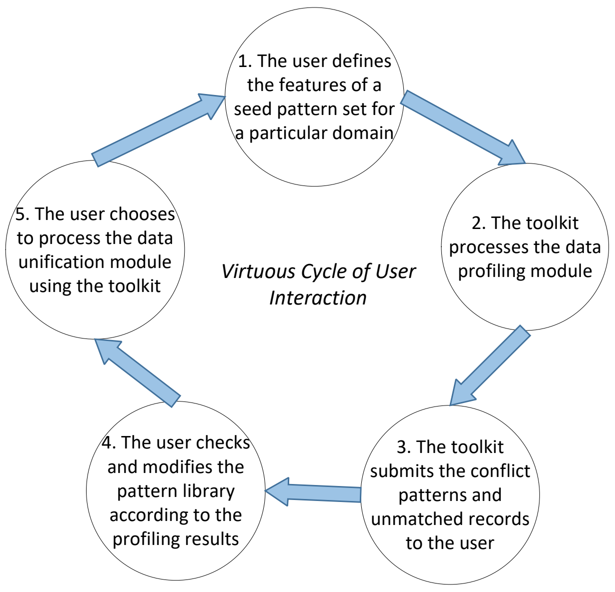

1. The user defines the features of a seed pattern set for a particular domain
2. The toolkit processes the data profiling module
3. The toolkit submits the conflict patterns and unmatched records to the user
4. The user checks and modifies the pattern library according to the profiling results
5. The user chooses to process the data unification module using the toolkit
Virtuous Cycle of User Interaction

FIGURE 3.2: The virtuous cycle of iterative and interactive approach.

Figure 3.2 describes the virtuous cycle with modules (data profiling and data unification) in the pattern based framework. This virtuous cycle incorporates five stages, which are ultimately implemented using a pattern based framework. The five stages are processed as follows. First, the user defines the features of patterns for the domain, and then the toolkit processes the data profiling in order to submit the conflict patterns and unmatched records to the user. The user will check the submitted results and modify the patterns to make the patterns closer to complete and mutually exclusive of each other. After that, the user may choose to do unification according to the submitted results. Or the user could continue to modify and define the pattern features in the toolkit to create the most appropriate pattern library for the particular domain. The different profiling results guide users in the virtuous cycle as follows:

- Unmatched records: Records appeared in this list means that the current pattern library could not cover these records. There are three situations. 1)The current pattern library is not complete, and we need to insert more patterns according to the unmatched records. 2).There are some characters which should includes in the current regular expressions. We need edit the regular expressions of current patterns. 3)There are some typos in the records which include an error character. It could be marked as an error of the records and should be edited by users.

- Conflict Records: Records appeared in this list matched with more than one patterns. A more restrict regular expression should be provided for the conflict patterns such as shrinking the scope of character.

3.3.2 Rules of pattern design

Pattern design constructs a pattern library $\mathbb{P}$ for each data domain. As discussed in Chapter 1, an ideal pattern library should be complete and mutual exclusive, in order to cover the entire dataset and eliminate pattern conflict. This incurs extensive efforts if we manually build the pattern library from scratch. We adopt an iterative and interactive approach to reduce human efforts.

Starting with a seed set of patterns based on the techniques for writing regular expressions in [24], we conduct pattern detection on a given dataset with the seed set, and obtain the *consistency profiling result* which contains information about unrecognised data records and conflicting patterns. Based on the consistency profiling result, we design and add more patterns into the seed set and meanwhile revise conflicting patterns, and then repeat the above process. We propose some heuristic rules for designing and revising patterns:

- Rule for fields: At each iteration, we revise the regular expression with the smallest coverage of records for each field. Consider the example in Table 1.4. We first apply the regular expression $(\,|[A{-}Za{-}z0{-}9])^{+}$ to the stop description field and then examine the unmatched records. We find there are still some other characters, such as '.','$\&$', and '$/$', occurring in the stop description field. Hence, we add these characters into $(\,|[A-Za-z0-9])^{+}$ and obtain a new regular expression $(\,|[A-Za-z0-9/\&.])^{+}$ for stop descriptions.

- Rule for separators: Based on the unmatched records, we discover new separators and add more patterns using different sequences of fields and separators. For example in Table 1.4, after finding new separators of "St", '-' and blank space, we include patterns such as "*D* - St *SN* [*ID*]", "*D* St *SN* [*ID*]","St *SN* *D* [*ID*]", etc.

- General rule: We assign higher priority to restricting the regular patterns rather than enlarging the coverage of regular expressions or reducing conflicting records. The balance between complete and conflict is a challenge in our framework design. The profiling results in virtuous cycle can generate a list of conflict data to users as a guide for further edit to patterns to detect and reduce the conflict. Also, the report of unmatched data could also guide user to find out more patterns in this domain. However, there may be a problem when one pattern is contained by another. If a pattern p_a could cover another pattern p_b, and only the p_a in the pattern library, the profling result will not show the records belongs to p_b. This will lead to mistakes in the next process. More patterns can reveal the mistake of one pattern covering another. In the example of Table 1.4, although the pattern "*D SN* [*ID*]" can match all the records mapped to the pattern "*D* Stop *SN* [*ID*]", we still include the pattern "*D* Stop *SN* [*ID*]" into the pattern library to guarantee we can recognise records matched to it.

3.4 Pattern Detection

String pattern matching has been researched for many years in different research fields such as information retrieval [56] and deep packet inspection [31]. As explained in Chapter 2, pattern detection in this framework could be considered as multiple regular expression pattern rules matching. However, in our context, the matching results will not be type boolean, while regular expression rules that each string could matched with is required. In addition, in relational database, regular expression rules matched with a set of strings instead of a long text string.

To handle m regular expression rules and find out all the possible matched regular expressions, two choices are possible: The **naive solution** is processing each strings individually in m automaton, or compiling different regular expression rules into a single automaton. Theoretically, the processing complexity of the naive solution is $O(nm)$ if each automaton is a DFA. On the other hand, when combining different regular expression rules into a single automaton, the complexity in worst case is also $O(nm)$ (the worst case refers to each string could be matched with every regular expression rules). Efforts were made by grouping states in DFA which could reduce the time complexity [53][55], however, these methods in literature could only deal with the string that could be matched with one regular expression, which is not our objective.

Regular expressions represent infinite set of patterns. We use the idea of FSM in our method. The difficulty is how to find all the matched patterns because of many-to-many mapping. The process of our pattern detection method is shown in Figure 3.3. In our method, we first construct a NFA(Nondeterministic Finite Automaton) for each pattern using Thompson's construction algorithm [48]. Since a pattern consists of fields(represented by regular expressions) and separators, we use series and parallel connections to construct our FSM. The FSM is a kind of NFA. Due to different principles of operations between DFA and NFA [1], using DFA for pattern detection is more efficient when doing matching process. However, since we need to detect all the possible patterns, we introduce a branch state for backtracking with pattern information to the standard DFA. This breaks the equivalence of NFA and DFA under our objective. In our approach, the FSM is optimized using subset construction algorithm [1] to achieve the DFA with backtracking. We introduce a break-point operation to the original subset construction algorithm. After the optimization of FSM, we could parse strings using the FSM. The details of the algorithms and evaluations of the method will be described in Chapter 5.

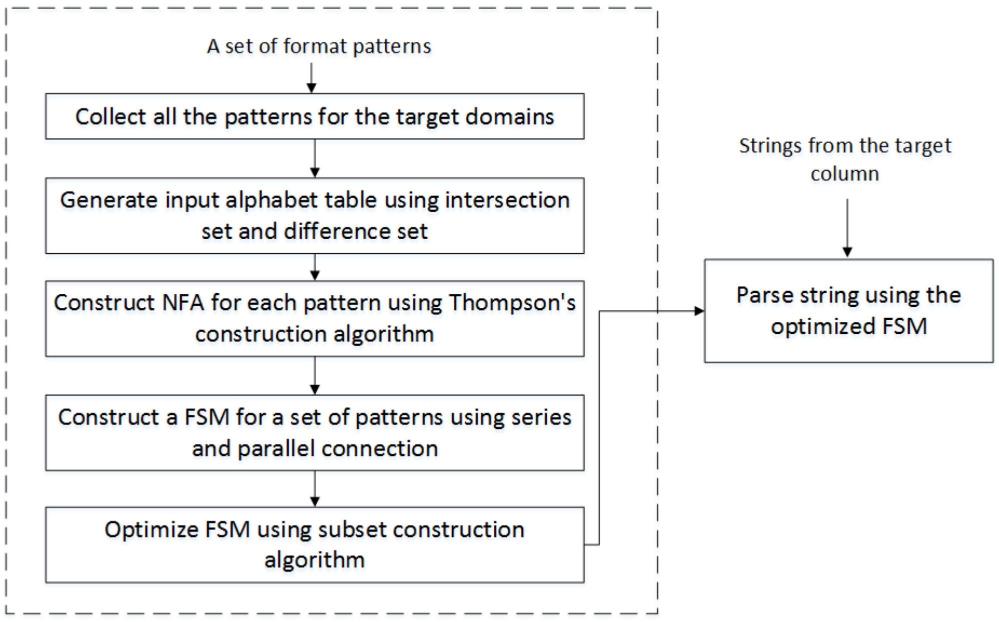

A set of format patterns
Collect all the patterns for the target domains
Generate input alphabet table using intersection set and difference set
Construct NFA for each pattern using Thompson's construction algorithm
Construct a FSM for a set of patterns using series and parallel connection
Optimize FSM using subset construction algorithm
Strings from the target column
Parse string using the optimized FSM

FIGURE 3.3: The process of pattern detection.

3.5 Pattern Unification

Given a dataset $\mathbb{D}$ from a specific domain along with the detected pattern for each data record $p_d \in \mathbb{P}_d$, pattern unification transforms all data records into a target pattern p^*. As discussed in

Chapter 1, pattern unification is more complicated than traditional string-based functions since it requires additional knowledge. In the example of “Alison St - St 32 [BT002904]” in Table 1.4, we need the semantic knowledge that the former “St” is part of the stop description filed while the latter “St” is a separator, in order to unify this data record to “Alison St - Stop 32 [BT002904]”. Consider “2003/03/17” in Figure 3.4 as another example. Both the semantic knowledge that the second “03” belongs to the month field and the domain knowledge that “03” as a month can be represented as “Mar” are necessary to transform “2003/03/17” to “2003-Mar-17”. To this end, we adopt a two-level pattern definition, namely a pattern is a sequence of fields and separators (semantic level) expressed as regular expressions (lexical level), and propose a split-transform-merge approach for pattern unification. Figure 3.4 illustrates an example of pattern unification.

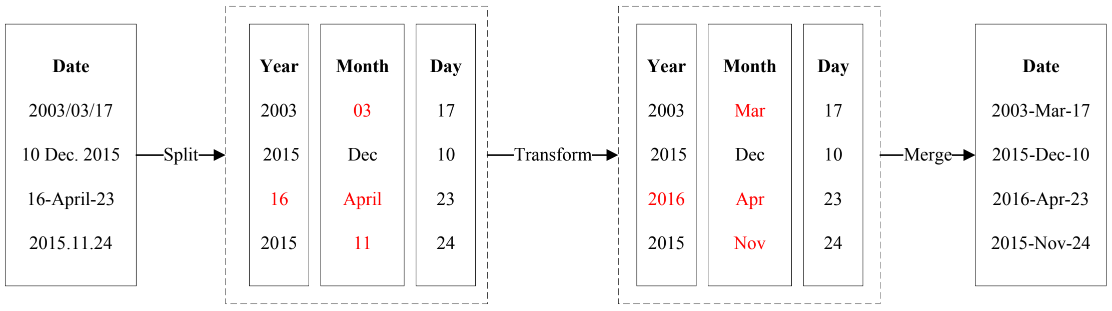


FIGURE 3.4: An example of pattern unification.

Split. As discussed in Section 3.4, our approach to pattern detection can not only determine matching patterns but also recognise boundaries between fields and separators. Hence, we split each data record from a specific domain into a set of field values and separator values, based on the pattern detection result. In Figure 3.4, each date (e.g., “2003/03/17”) is divided into three fields namely year (e.g., “2003”), month (e.g., “03”), and day (e.g., “17”). For ease of representation, we ignore the separators in Figure 3.4.

Transform. Given the set of field values and separator values, we conduct transformation according to the target pattern p^*. We introduce transformation functions for this task, many of which cannot be easily supported by traditional string-based functions. Table 3.1 presents some examples of our proposed transformation functions which can be roughly classified into two categories: structure change and value change. Structure change functions reorganise the order of fields and separators, while value change functions modify the actual values of fields and separators. We associate domain knowledge with some value change functions such as fielding mapping, abbreviation, etc. In Figure 3.4, the month field value “03” is modified to “Mar”.

Merge. After transformation, we merge the revised values of fields and separators together using string concatenation function to obtain the unified data record. In Figure 3.4, the pattern unification result for data record “2003/03/17” is “2003-Mar-17”.

TABLE 3.1: Examples of transformation functions.

Category	**Functions**	**Examples**
Structure Change	field order change	“John Stevens” → “Stevens John”
Value Change	separator change	“2014/03/13” → “2014-03-13”
	field mapping	“03” → “Mar”
	measurement change	“10cm” → “0.1m”
	abbreviation	“Stevens John” → “Stevens J”
	string reverse	“abc” → “cba”

3.6 Summary

In this chapter, we propose an user-driven pattern-based framework including pattern design, pattern detection, and pattern unification modules. For pattern design in SQL extension, we propose a virtuous cycle of iterative and interactive approach and rules of pattern design. We introduce a modification to classic string pattern matching technology especially DFA technology to ensure the efficiency and accuracy of the pattern detection module. The details about the modified DFA and the evaluation of the method will be presented in Chapter 5. For pattern unification module, we employ split-transform-merge method to unify the inconsistent records.

Chapter 4

Design of SQL Extensions

4.1 Introduction and Motivation

As noted in Chapter 1, classic integrity constraints have been used in most DBMSs to improve data quality. In general, each constraint is defined in a declarative way, which costs less than enforcing rules by using a large number of standard SQL statements [49]. Use of declarative SQL extensions is a popular way to address database problems which cannot be resolved using standard SQL (SQL:2011). SQL extensions are also employed in the data quality field. Helena et al. [25] propose an AJAX data cleansing framework using a declarative language based on five logical data transformation operators (mapping, matching, clustering, merging and view). These operators are represented using a detailed syntax which is in the spirit of SQL. Many other research works focus on applying SQL extensions to data quality problems, such as duplication detection [21]. For data inconsistency issues, [37] presents an SQL extension to deal with data inconsistency in heterogeneous information sources. Both integrity constraints and these SQL extensions focus on the relationships between tables, rows and columns, and seldom focus on records at the instance level including the format or structure of the data items. Therefore, we designed SQL extensions according to our pattern based framework to describe the features inside the tuples. In this chapter, we will introduce the grammar of the SQL extensions we designed and provide an example to show how these SQL extensions are used.

4.2 SQL Extensions

The most recent SQL standard SQL:2011 supports declarative integrity constraints. Our SQL extensions specifically focus on data representation consistency at an instance level based on SQL:2011. Our pattern-based framework that was introduced in chapter 3, comprises three parts: definition, detection, and unification. The SQL extensions we designed cover a) the declarative definition of the patterns, and b) the operations to improve data representation consistency. The latter operators include detection, and unification.

4.2.1 Declarative definition of patterns

According to the two-level pattern definition in Chapter 3, we design the definition of patterns in SQL extension based on the lexical level and semantic level.

- Lexical level - using "Create field" statements to define the structure features of each field in patterns. The structure features are represented by regular expressions.
- Semantic level - using "Create format pattern" statements to define the sequence of fields and separators. The separators are represented by regular expressions.

Hence, five new clauses and three keywords, namely **FIELD**, **REG**, **FORMAT PATTERN** and **MAPPING**, should be added to the syntax of the SQL standard. There are four main extended clauses:

1. Create field:
   CREATE FIELD $[< domain >]$ $< field\_name >$ **REG** $regular\_expression >$
   [**MAPPING** $< mapping\_list > < values\_or\_query\_spec >$];
   Note: The field name should be unique. If there is a mapping in the field, the mapping list should be complete, that is, all the values in that field should have their mapping values. Furthermore, there should be a one-to-one correspondence between mapping name and value.

2. Create format pattern:
   CREATE $< entity\_classes >$ **FORMAT_PATTERN** $< pattern\_representaion\_list >$;
   Note: All the fields appearing in the pattern representation list should be defined before

they appear in this list. Each pattern representation should follow the rules for the design of patterns in Chapter 3. This means that each pattern consists of the defined field name and separators which are represented by regular expressions.

3. Delete domain:

   DELETE FIELD $< field\_name >$;

   Note: When a field is deleted, the related mappings and patterns will be deleted automatically.

4. Delete format pattern:

   DELETE FORMAT_PATTERN $< pattern\_representation\_list >$;

The BNF syntax of the four new clauses are as follows:

$< \textbf{field\_name} >$ ::= **NAME**;

$< \textbf{mapping\_list} >$ ::= $< mapping\_name >$

$\quad\quad | < mapping\_list > < mapping\_name >$

$< \textbf{mapping\_name} >$ ::= **NAME** $< opt\_type >$;

Note: The default type of mapping value is string.

$< \textbf{entity\_classes} >$ ::= **NAME**;

$< \textbf{pattern\_representation\_list} >$ ::=

$\quad\quad < pattern\_representation >$

$\quad\quad | < pattern\_representation\_list > < pattern\_representation >$

$< \textbf{pattern\_representation} >$ ::= **STRIING** $< opt\_seperators >$

$< \textbf{opt\_seperators} >$ ::=

$\quad\quad$ | **STRING**;

4.2.2 Detection Operation:

A simple SQL extension is designed to detect the patterns of the records. One new clause and a keyword, namely **PROFILE**, should be added referring to the syntax of the SQL standard. There are four main extended clauses:

PROFILE $< domain >$ $< column\_name >$

FROM $< table\_ref\_list >$

The BNF syntax of new clause is as following:

$<$ **profile_clause** $>$ ::=

PROFILE$< domian >$ $< column\_name >$ **FROM** $< table\_ref\_list >$

4.2.3 Unification Operation:

In order to unify the different formats, one new optional clause $< reformat\_clause >$, a new $< search\_condition >$ for formats and a key word **FORMAT_PATTERN** are added to the standard SQL. The statement of the new "unify" clause is based on the "update" statement.

UPDATE $< table\_ref\_list >$

SET col_name1 $< reformat\_clause >$ [,col_name2 = expr1,...]

[**WHERE** $< search\_condition >$] [**ORDER BY**...][**LIMIT** row_count]

The BNF syntax of the new clause is as followings:

$<$ **reformat_clause** $>$ ::=

FORMAT_PATTERN = $< format\_identifying >$

$<$ **format_identifying** $>$ ::= INTEGER | NAME

Note: $< format\_identifying >$ should be the format id or the name which appeared in the format dataset.

$<$ **search_condition** $>$ ::=

$< format\_condition\_list >$

| $< search\_condition >$ **OR** $< search\_condition >$

| $< search\_condition >$ **AND** $< search\_condition >$

| **NOT** $< search\_condition >$

| $< predicate >$

| ...

$<$ **format_condition_list** $>$ ::=

$$< format\_condition >$$

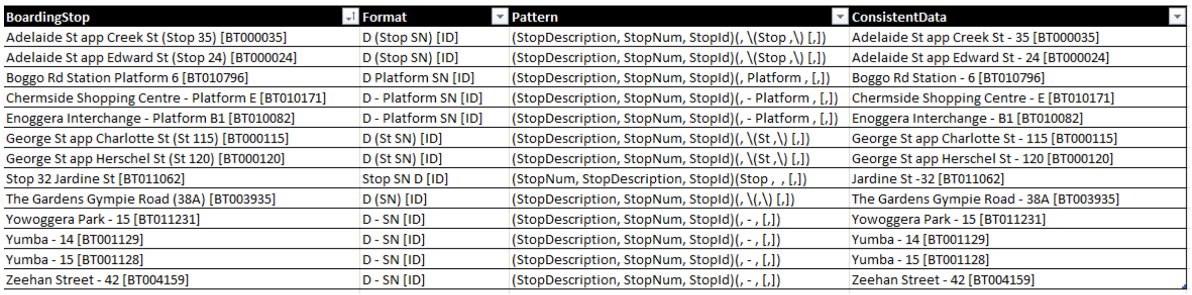
$$|\ < format\_condition\_list > < format\_condition >$$

$< \textbf{format\_condition} > ::=$

$< column\_ref >$ **FORMAT_PATTERN** $= < format\_identifying >$

4.3 Usages and Examples

The representation of attribute “BoardingStop” in gocard dataset is inconsistent due to the system version and some typing errors. The example in Figure 4.1 shows seven different stop formats in attribute “BoardingStop”. We use this example to explain the usage of our SQL extensions to address this inconsistency problem in “BoardingStop”.



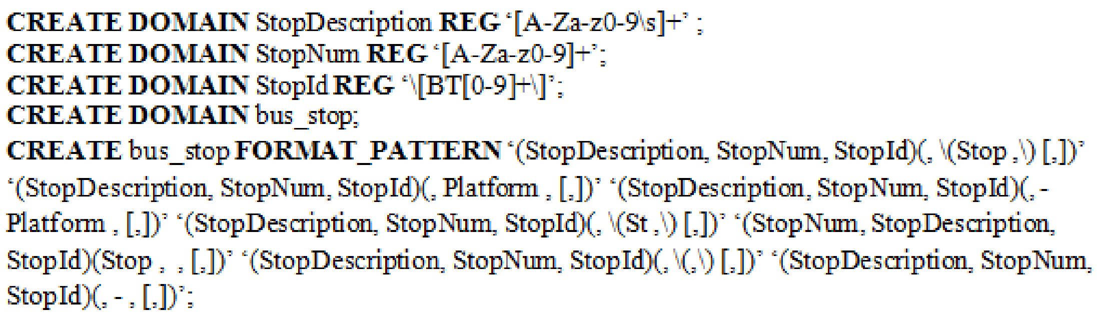
BoardingStop	Format	Pattern	ConsistentData
Adelaide St app Creek St (Stop 35) [BT000035]	D (Stop SN) [ID]	(StopDescription, StopNum, StopId)(, \(Stop ,\) [,])	Adelaide St app Creek St - 35 [BT000035]
Adelaide St app Edward St (Stop 24) [BT000024]	D (Stop SN) [ID]	(StopDescription, StopNum, StopId)(, \(Stop ,\) [,])	Adelaide St app Edward St - 24 [BT000024]
Boggo Rd Station Platform 6 [BT010796]	D Platform SN [ID]	(StopDescription, StopNum, StopId)(, Platform , [,])	Boggo Rd Station - 6 [BT010796]
Chermside Shopping Centre - Platform E [BT010171]	D - Platform SN [ID]	(StopDescription, StopNum, StopId)(, - Platform , [,])	Chermside Shopping Centre - E [BT010171]
Enoggera Interchange - Platform B1 [BT010082]	D - Platform SN [ID]	(StopDescription, StopNum, StopId)(, - Platform , [,])	Enoggera Interchange - B1 [BT010082]
George St app Charlotte St (St 115) [BT000115]	D (St SN) [ID]	(StopDescription, StopNum, StopId)(, \(St ,\) [,])	George St app Charlotte St - 115 [BT000115]
George St app Herschel St (St 120) [BT000120]	D (St SN) [ID]	(StopDescription, StopNum, StopId)(, \(St ,\) [,])	George St app Herschel St - 120 [BT000120]
Stop 32 Jardine St [BT011062]	Stop SN D [ID]	(StopNum, StopDescription, StopId)(Stop , , [,])	Jardine St -32 [BT011062]
The Gardens Gympie Road (38A) [BT003935]	D (SN) [ID]	(StopDescription, StopNum, StopId)(, \(,\) [,])	The Gardens Gympie Road - 38A [BT003935]
Yowoggera Park - 15 [BT011231]	D - SN [ID]	(StopDescription, StopNum, StopId)(, - , [,])	Yowoggera Park - 15 [BT011231]
Yumba - 14 [BT001129]	D - SN [ID]	(StopDescription, StopNum, StopId)(, - , [,])	Yumba - 14 [BT001129]
Yumba - 15 [BT001128]	D - SN [ID]	(StopDescription, StopNum, StopId)(, - , [,])	Yumba - 15 [BT001128]
Zeehan Street - 42 [BT004159]	D - SN [ID]	(StopDescription, StopNum, StopId)(, - , [,])	Zeehan Street - 42 [BT004159]

FIGURE 4.1: Example of “BoardingStop” in goCard data.

1. **Setting** As we can see, bus stop is not a commonly used field. We need to set the possible format and create a pattern libary for bus stop which could be used not only for “BoardingStop” but “AlightingStop”. In this example, we set the field features which is represented by regular expressions, and then set the pattern features which consist of field names and separators. The SQL script is shown in Figure 4.2.

2. **Detection** After setting the pattern library for bus stops, we could detect the format for each “BoardingStop” data. This operator helps users understand the inconsistency problems in the target dataset. The result of this operation is the possible formats for each tuple in “BoardingStop”. The SQL statement is as follows:

   PROFILE bus_stop BoardingStop **FROM** gocard records;



```
CREATE DOMAIN StopDescription REG '[A-Za-z0-9\s]+' ;
CREATE DOMAIN StopNum REG '[A-Za-z0-9]+';
CREATE DOMAIN StopId REG '\[BT[0-9]+\]';
CREATE DOMAIN bus_stop;
CREATE bus_stop FORMAT_PATTERN '(StopDescription, StopNum, StopId)(, \(Stop ,\) [,])'
'(StopDescription, StopNum, StopId)(, Platform , [,])' '(StopDescription, StopNum, StopId)(, -
Platform , [,])' '(StopDescription, StopNum, StopId)(, \(St ,\) [,])' '(StopNum, StopDescription,
StopId)(Stop , , [,])' '(StopDescription, StopNum, StopId)(, \(,\) [,])' '(StopDescription, StopNum,
StopId)(, - , [,])';
```

FIGURE 4.2: Example: Setting operation

3. **Unification** This opearation will update the data to satisfy data representational consistency. In this stage, we need to set a target format, and the result is the consistent BordingStop data shown in Figure 4.1 which follows the format of '(StopDescription, StopNum, StopId)(, - , [,])'. The SQL statement as follows:

   UPDATE gocardrecords **SET** BoardingStop
   FORMAT_PATTERN = '(StopDescription, StopNum, StopId)(, - , [,])';

4.4 Summary

In conclusion, we designed SQL extensions for data representation consistency, and employ an example to show how these extensions can be used to achieve data representation consistency. Compared with integrity constrains in SQL:2011, which need detailed constraint information, and without running a large number of update operations, our SQL extensions reduce the knowledge and skill requirements from the user end, update inconsistent records to achieve consistent data and provide the capability to search inconsistent data. Integrity constraints are specified within a create table or alter table statement and take affect when new records are inserted or updates are performed, whereas the SQL extensions in this thesis target the existing dataset which needs to be repaired.

Chapter 5

A Modified DFA Method for Data Inconsistency Pattern Detection

5.1 Introduction and Motivation

Data inconsistency detection and profiling is a crucial task in checking and ensuring the quality of data. Historically, the methods to detect and resolve data inconsistency problems were developed with respect to a certain context, and rule-based methods are employed to deal with a general context. Recently, CFDs (Conditional functional dependencies) were introduced to capture data inconsistencies by enforcing bindings of semantically related values, and rule-based methods for data inconsistency problems, such as editing rules and fixing rules, are based on CFDs. However, these rule-based methods employ the dependencies between columns for every tuple to monitor and repair semantic data inconsistencies. There is a lack of domain agnostic methods for data representation inconsistency detection that will cover each cell for each cell in a dataset.

In this chapter, we introduce a DFA (Deterministic Finite Automaton) based method to detect all possible patterns for each record and generate the profiling result for the target data. Since the traditional DFA methods are not available in our context, we introduce a modification to the traditional DFA method. Regular expression matching using deterministic finite automata (DFA) is a well-studied problem in both theoretical and practical settings. However, as noted in Chapter 1, the classic method will not fit our context. In this thesis, we provide a modified DFA with labels and a back-tracking process to achieve that additional requirement and to evaluate both the efficiency and accuracy of the matching engine. In this chapter, the technical details about our modified DFA approach will be presented. Then, our evaluation of this approach will verify both the efficiency and accuracy of the method using real world datasets.

5.2 Approach

With the pattern library $\mathbb{P}$ at hand, pattern detection needs to recognise the possible patterns $\mathbb{P}_d$ for each data record $d \in \mathbb{D}$ from a specific domain. A straightforward approach is to check every record-pattern pair one by one to see whether the record conforms to the pattern. The time complexity of such a task is obviously $O(|\mathbb{D}| \times |\mathbb{P}|)$, where $|\mathbb{D}|$ and $|\mathbb{P}|$ denote the size of the dataset and the pattern library, respectively. In this work, we improve the efficiency by conducting pattern detection in a batch process.

As defined in Section 3.1, a pattern is a sequence of fields and separators which are represented by regular expressions. Therefore, a pattern can also be regarded as a concatenation of all the regular expressions. In order to determine whether a data record (i.e. a string) satisfies a pattern, it is suffice to check whether this record can be recognised by the concatenated regular expression. This is typically accomplished using a Finite State Machine (FSM) such as a Nondeterministic Finite Automaton (NFA) and a Deterministic Finite Automaton (DFA). Figure 5.2 illustrates the process of constructing an NFA to represent a pattern based on Thompson's construction algorithm [48]. According to [48], there are five basic rules during NFA construction as shown in Figure 5.1 ($N(s)$ and $N(t)$ is the NFA of the subexpression s and t, respectively).

We first build NFAs for each field and separator based on their regular expressions using the five rules in Figure 5.1, and then combine these NFAs using a *series connection*. In order to detect patterns via a batch process, we compile the entire pattern library into one NFA such that all possible patterns for a data record can be detected by scanning the record only once. Specifically, we employ parallel connection to combine the NFAs for each pattern into a large NFA.

In the following we consider a simple pattern library as a running example. The field set and the separator set are $\mathbb{F} = \{f_1 = [ab1]^+, f_2 = [a1]^+\}$ and $\mathbb{S} = \{s1 = 11, s2 = a1\}$, respectively, and the pattern library is $\{p_1 = f_1 s_1 f_2 = [ab1]^+ 11 [a1]^+, p_2 = f_1 s_2 f_2 = [ab1]^+ a1 [a1]^+\}$. Figure

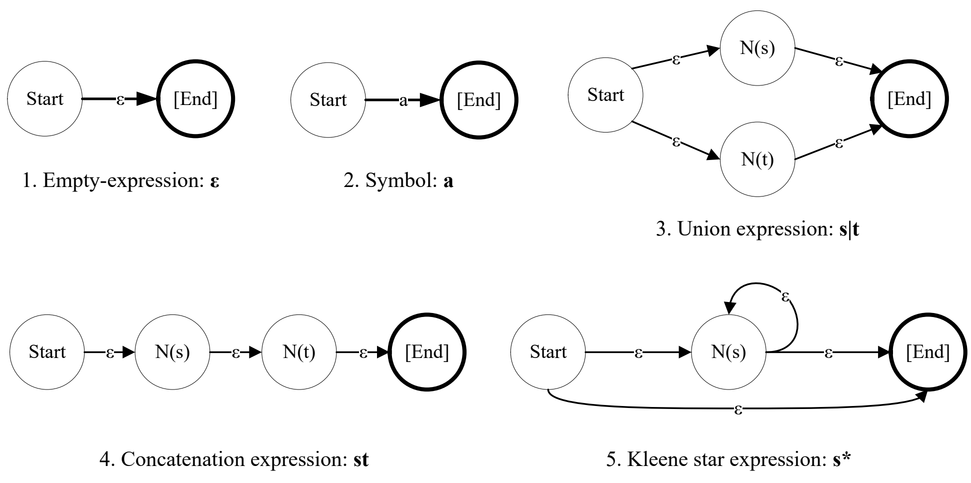


FIGURE 5.1: Five rules in Thompson's algorithm.

5.3 illustrates an NFA for this pattern library. In order to split each data record (e.g. "aba11a") into a collection of field values (e.g. "aba" and "a") and separator values (e.g. "11"), which is a prerequisite for pattern unification and as discussed in Section 3.5, pattern detection should be able to determine both the matching patterns and the boundaries between fields and separators. Hence, we assign each final state in the NFA with a special label which is used for notification of the pattern that is detected upon reaching the final state. We also assign each state corresponding to a separator with a special label (i.e. yellow states in Figure 5.3) to differentiate fields and separators.

The NFA for a pattern library is usually huge, containing thousands of states with multiple choices of transitions. To reduce computational cost when processing the NFA, we need to ensure the constructed NFA has as few states as possible. A classic approach is to transform the original NFA into a DFA using a subset construction algorithm and then minimise the DFA [3]. Figure 5.4 shows the minimised DFA of the original NFA in Figure 5.3. There are three obvious drawbacks of the classic DFA: 1) it has no back-tracking and hence cannot recognise multiple patterns for a data record (e.g. "aba11a" $\rightarrow \{p_1, p_2\}$,"abba1a" $\rightarrow \{p_2\}$); 2) it cannot determine which pattern the data record maps to when reaching the final state; and 3) it cannot detect boundaries between fields and separators. The key problem is that classic subset construction and DFA minimisation algorithms combine and reduce states without considering their specific meanings, namely whether the states are fields or separators and to which pattern the states correspond. Hence, we introduce several

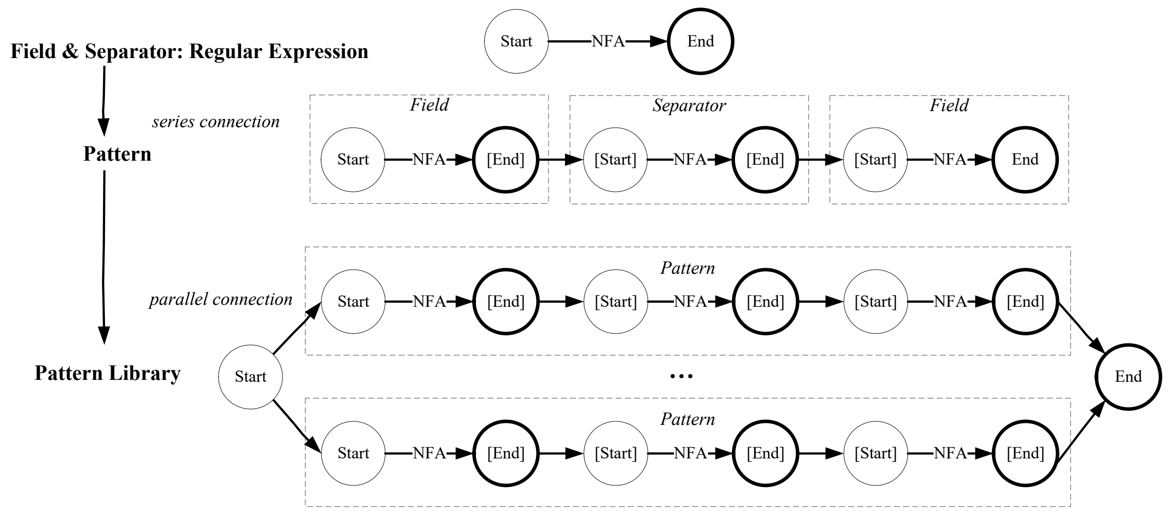


FIGURE 5.2: Construction of an NFA to encode the pattern library.

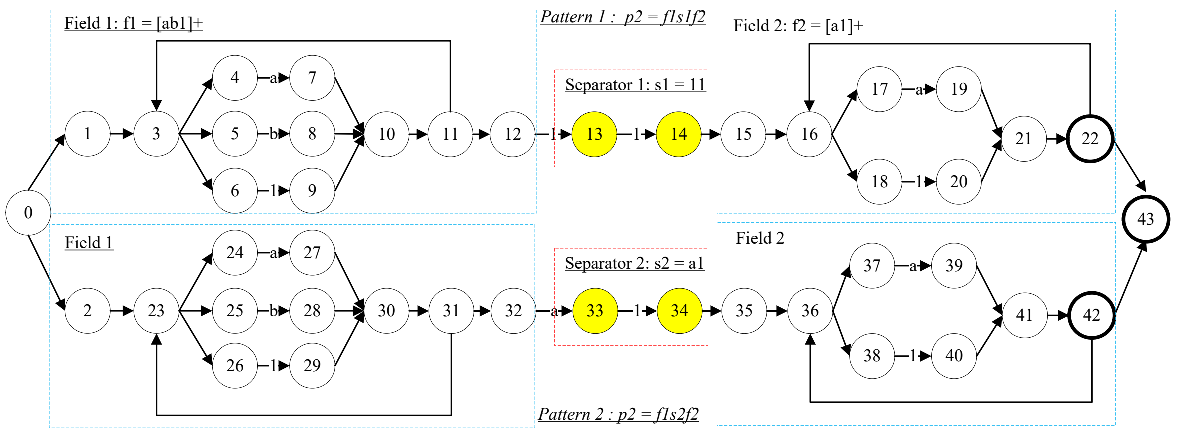


FIGURE 5.3: NFA for $\{[ab1]^{+}11[a1]^{+}, [ab1]^{+}a1[a1]^{+}\}$.

modifications to the subset construction and DFA minimisation algorithms (as shown in Algorithm 1). Specifically, in the subset construction algorithm we not only introduce state labels but also introduce a "break" operation (as shown in Figure 5.5) compared with the classic subset construction algorithm. This operation separates the states by different patterns to avoid a misleading pattern matching result.

One requirement of the modified DFA is to recognise multiple patterns by scanning the data record only once. In other words, after recognising one pattern, the modified DFA should enable back-tracking to check other patterns. Back-tracking can be easily supported by NFA since NFA allows for multiple next states given a specific input symbol. However, this is not the case for

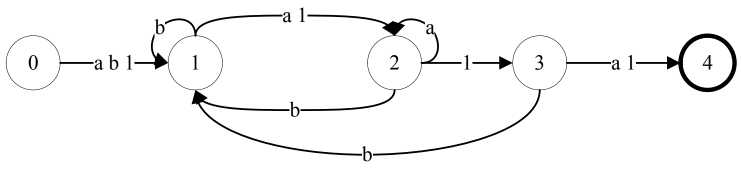
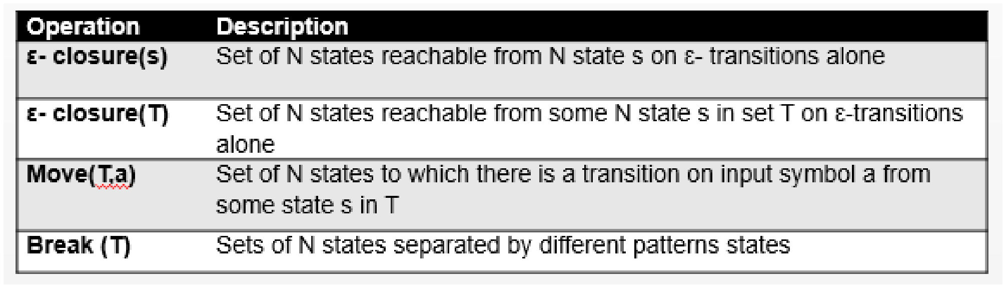


FIGURE 5.4: Classic DFA for $\{[ab1]^{+}11[a1]^{+}, [ab1]^{+}a1[a1]^{+}\}$.



Operation	Description
ε- closure(s)	Set of N states reachable from N state s on ε- transitions alone
ε- closure(T)	Set of N states reachable from some N state s in set T on ε-transitions alone
Move(T,a)	Set of N states to which there is a transition on input symbol a from some state s in T
Break (T)	Sets of N states separated by different patterns states

FIGURE 5.5: Operations of the modified subset construction algorithm.

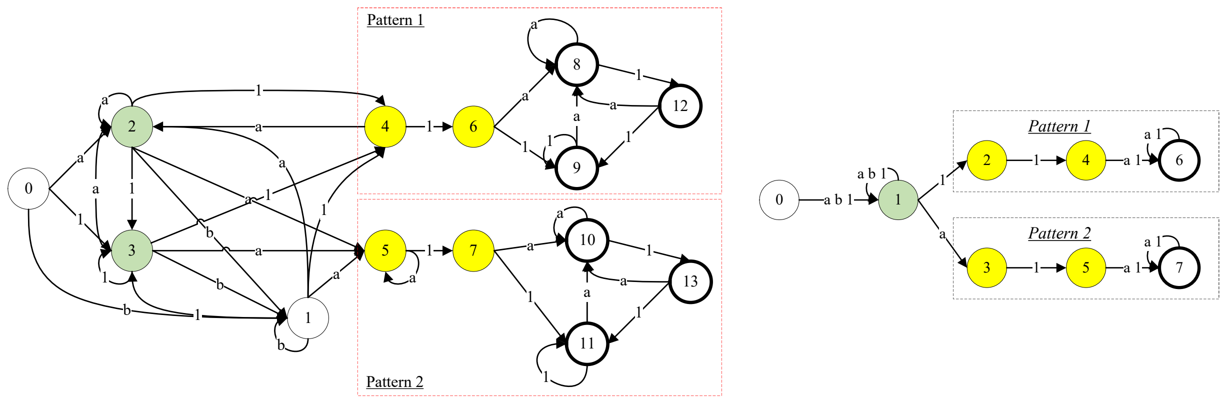
DFA. Therefore, we introduce a special state-branch state-into the modified DFA which can support multiple transitions on a single input symbol. Furthermore, in order to distinguish patterns, the information that different states lead to different patterns should be retained in the modified DFA. To this end, we first check which patterns the states lead to when merging states in subset construction. If the states represent multiple patterns, we then combine them into a single branch state and add a transition for each pattern. Branch states will not be merged with other states in DFA minimisation. The automatons in Figure 5.6 show modified DFAs of the NFA in Figure 5.3. The green states are branch states which transit to two next states corresponding to patterns p_1 and p_2, respectively.

Another requirement for the modified DFA is to determine patterns and pattern boundaries. As mentioned above, we assign states in the NFA with two special labels, i.e. a pattern label and a separator label, to notify of matching patterns and boundaries between fields and separators, respectively. This information should be retained in the minimised DFA. In particular, one constraint is that separator states as well as final states corresponding to different patterns cannot be merged with each other or with other normal states when conducting subset construction and DFA minimisation. Figure 5.6(a) and Figure 5.6(b) demonstrate the modified subset construction and minimisation of the NFA in Figure 5.3, whereby the separator states and final states are retained.

Algorithm 1: Subset construction algorithm of optimizing FSM

```
Require:
    A FSM $N$ with pattern marks in each states;
Ensure:
    An optimized FSM $F$ could match the same pattern set as $N$;
1:  initially, $\epsilon - closure(s_0)$ is the only state in $Fstates$, and it is unmarked;
2:  while there is an unmarked state $T$ in $Fstates$ do
3:      mark $T$;
4:      for each input symbol $a$ do
5:          subset of $F$ states $U := \epsilon - closure(move(T, a))$;
6:          if $u_i \notin Fstates$ then
7:              add a transition $Ftran(T, a) := U$ in $N$;
8:      if the states in $U$ comes from different patterns then
9:          mark $U$ as a branch state;
10:         Dividing $U = u_1, u_2, .., u_k$ according to different patterns;
11:         for Each $u_i \in U$ do
12:             if $u_i \notin Fstates$ then
13:                 add $u_i$ as an unmarked state to $Fstates$ in $N$;
14: return $N$;
```

Given the modified DFA for a pattern library, we can recognise matching patterns for a data record accurately and efficiently. Consider the modified DFA in Figure 5.6(b) and the data record “aba11a” as an example. At branch state 1, the automaton can go through states $1 \rightarrow 1 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 6$ to arrive at pattern p_1 (with “aba” and “a” as the field values, and “11” as the separator value), but it can also back-track and go through states $1 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 7$ to arrive at pattern p_2 (with “ab” and “1a” as the field values, and “a1” as the separator value). The back-tracking ends when there is no next state for the branch state or the last character of the data record can not reach a final state.



(a) Modified subset construction. (b) Modified DFA minimisation.

FIGURE 5.6: Modified DFA for $\{[ab1]^{+}11[a1]^{+}, [ab1]^{+}a1[a1]^{+}\}$.

5.2.1 Method Evaluation

We conducted extensive experiments to evaluate the accuracy and efficiency of our framework for addressing data representational inconsistencies. All the algorithms were implemented in C#, and all the experiments were conducted on a server with 2.90 GHz Intel Xeon E5-2690 CPU and 192 GB memory.

Our evaluation was based on real-life datasets from various domains including public transportation data[1], and DBLP data[2]. Specifically, we downloaded a one-month (i.e. March 2013) snapshot of Translink gocard data in Brisbane, which consists of 4,329,128 records of gocard touch-on and touch-off information. We chose the noisy boarding stop domain to conduct our experiments. We also downloaded a DBLP snapshot on May 16, 2016, which contains 10,195,320 records of computer science bibliography information. For this dataset, we chose the inconsistent author name domain for our experiments.

5.2.2 Accuracy

Using the iterative and interactive approach to pattern designing, we constructed a pattern library of 18 patterns (Table 5.1) for the boarding stop domain, and 17 patterns (Table 5.2) for the author name domain.

[1]https://mobile.translink.com.au/about-translink/open-data
[2]http://dblp.uni-trier.de/xml/

TABLE 5.1: Pattern library for "BoardingStops" in the gocard data.

Pattern[*]				
D - *SN* [*ID*]	*D* (Stop *SN*) [*ID*]	*D* - Platform *SN* [*ID*]	*D* - Zone *SN* [*ID*]	*D* (St *SN*) [*ID*]
D - Stop *SN* [*ID*]	*D*-*SN* [*ID*]	*D* Platform *SN* [*ID*]	*D* Zone *SN* [*ID*]	*D* (stop *SN*) [*ID*]
D Stop *SN* [*ID*]	*D* (*SN*) [*ID*]	*D*, platform *SN* [*ID*]	*D* -*SN* [*ID*]	*D* stop *SN* [*ID*]
D '*SN*' [*ID*]	Stop *SN* *D* [*ID*]	*D* - Platform '*SN*' [*ID*]		

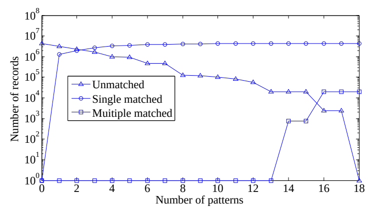
[*] Stop description: *D*= $(\vert[A-Za-z0-9/!,;' @\#\$\%\&*()-.”])^{+}$; Stop number: *SN*=$[A-Za-z0-9/]^{+}$; Stop ID: *ID*=$BT[0-9A-Za-z\_]^{+}$.

TABLE 5.2: Pattern library for authors in the DBLP data.

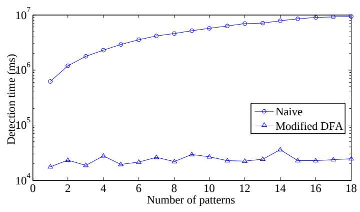
Pattern[*]				
G F	*G prefix F*	*G F suffix*	*G_abbr1 F*	*G_abbr1 prefix F*
G_abbr1 F suffix	*G_abbr2 F*	*G_abbr2 prefix F*	*G_abbr2 F suffix*	*F G_abbr1*
G (*O*) *F*	*G* (*O*) *prefix F*	*G* (*O*) *F suffix*	*G_abbr1* (*O*) *F*	*G_abbr1* (*O*) *prefix F*
G "*O*" *F*	*G_abbr2* (*O*) *F*			

[*] Given name: *G*=$(\vert[a-zA-Z'-])^{+}$, *G_abbr1*=$(\vert[A-Za-z.'-])^{+}.$, *G_abbr2*=$(\vert[A-Za-z.'-])^{+}.((\vert-)[a-zA-Z]^{+})^{+}$; Family name: *F*=$[a-zA-Z'-]^{+}$; Other name: *O*=$(\vert[a-zA-Z-])^{+}$;
prefix=bin|Di|Del|del|Della|Dalle|De|D|Da|da|Dal|Dall|Dalla|Dalle|di|del|de|da|Es|Du|el|es|de los| de las|de la|des|du|Von|Van|Van den|Van der|von|von der|Al|Au|al|am;
suffix=(Jr.|Sr.|I|II|III|IV|V)$^{+}$.

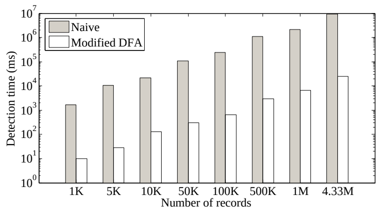
To obtain more insight into the performance and contribution of each pattern, we rank the patterns according to their coverage (i.e. number of matched records) in descending order, add them into the pattern library one by one, and then check the coverage and conflicts of the resulting pattern library. Figure 5.7(a) illustrates the change of in the number of unmatched records, single matched records and multiple matched records in log-scale when adding patterns into the library. We can see that the first pattern (i.e. *D* - *SN* [*ID*]) captures more than 25% of the boarding stops and the author names in the gocard data and DBLP data, respectively. With more patterns inserted into the pattern library, its coverage increases gradually, at the cost of more conflicts among patterns.



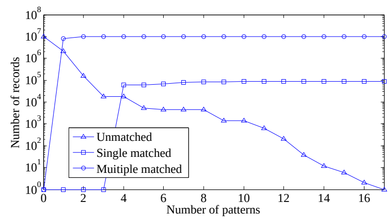
(a) Pattern accuracy in the gocard data.



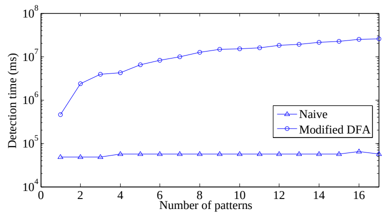
(b) Detection time with different library size in gocard data.



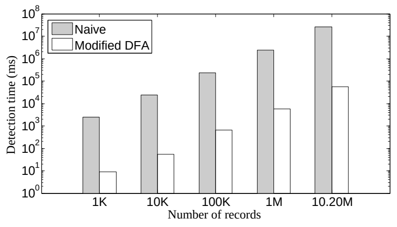
(c) Detection time with different dataset size in gocard data.



(d) Pattern accuracy in DBLP data.



(e) Detection time with different library size in DBLP data.



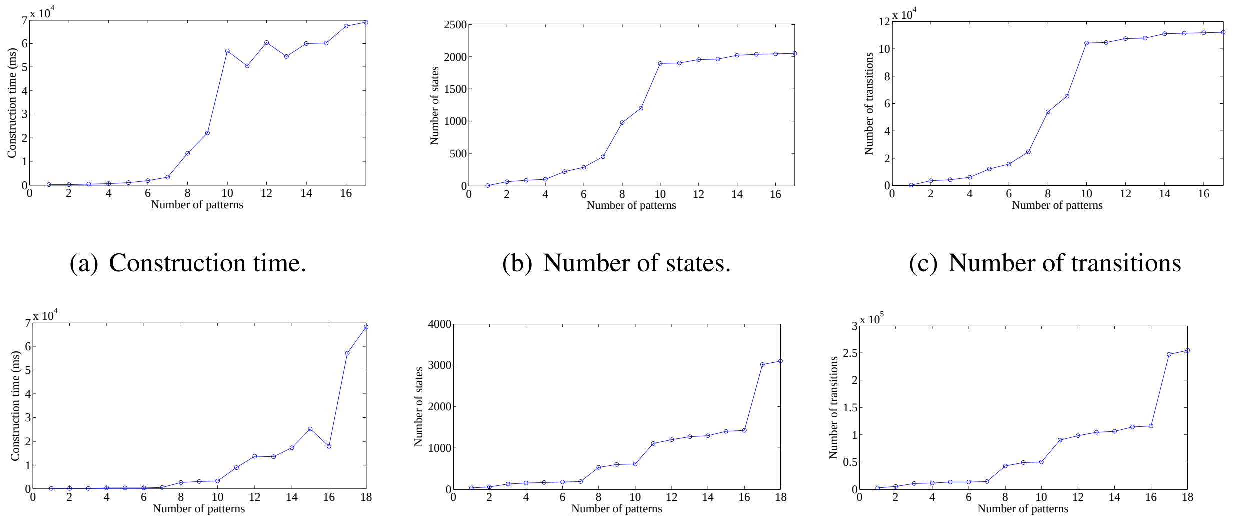
(f) Detection time with different dataset size in DBLP data.

FIGURE 5.7: Accuracy evaluation.

5.2.3 Efficiency

Given a pattern library and an inconsistent dataset, the naive solution for pattern detection is pair-wise checking. In order to improve efficiency, we propose to examine patterns via a batch process. Specifically, we combine all patterns into an NFA and then utilise modified subset construction and DFA minimisation algorithms to reduce the original NFA into a modified DFA. In this section, we compare the pattern detection time of the naive solution with that of the modified DFA. We change the size of the pattern library and the size of the dataset, and report the evolving pattern detection time for the gocard data in Figure 5.7(b) and Figure 5.7(c), and for the DBLP data in Figure 5.7(e) and Figure 5.7(f), respectively. As we can see, the pattern detection time for both methods increases linearly with the growth in size of the dataset. But when more patterns are included in the pattern library, the detection time of naive solution still rises linearly while our modified DFA remains stable.

Although DFA construction can be accomplished offline, it can still be quite time consuming especially when the pattern library is huge. Therefore, we also evaluate the DFA construction time when varying the size of the pattern library. From Figure 5.8(d) and Figure 5.8(a), we can see



(a) Construction time. (b) Number of states. (c) Number of transitions

(d) Construction time. (e) Number of states. (f) Number of transitions

FIGURE 5.8: Efficiency Evaluation.

that the DFA construction cost grows when the number of patterns increases. To obtain a more insightful understanding of the increase in construction time, we present the number of states and transitions in the DFA in gocard data respectively in Figure 5.8(e) (Figure 5.8(b) in DBLP data) and Figure 5.8(f) (Figure 5.8(c) in DBLP data). It is obvious that when the size of the pattern library increases, the resulting DFA becomes larger, and consequently the construction time increases.

5.3 Summary

In this chapter, we propose the modified DFA method for pattern detection and design experiments to evaluate both the accuracy and efficiency of the approach using gocard data and DBLP data. We introduce modifications to the classic subset construction and DFA algorithms to satisfy our requirements as discussed in Chapter 1. The modified DFA does not only report all possible patterns for each record, but detects each field in the record. In addition, the evaluation results using both the gocard data and the DBLP data illustrate the accuracy and efficiency improvement compared to the naive approach.

Chapter 6

Implementation of Toolkit

The toolkit for improving data representational consistency with SQL extensions was implemented during my Mphil candidate period. This toolkit includes three main modules based on the framework proposed in Chapter 3, namely, a) the SQL extension parser, b) the inconsistency profiler, and c) the inconsistent data unifier. In this chapter an overview of the system architecture is presented, followed by the subsystem design for each module and the limitations of regular expression patterns. Finally a use case is used to describe the usage of this toolkit.

6.1 System Architecture

All the algorithms introduced in Chapter 3 were implemented in C++ with MFC (Microsoft Foundation Classes) for the user interface, and compiled using Visual Studio 2012 connected to a MySQL database. The system overview is shown in Figure 6.1 and it has three tiers, namely, the view tier, the control tier, and the model tier.

- View tier – directly related to the functional requirements and user interface. The MFC framework is employed to interact with the users.
- Control tier – includes actions and processing components for each requirement.
- Model tier – represents the data connections with the MySQL database.

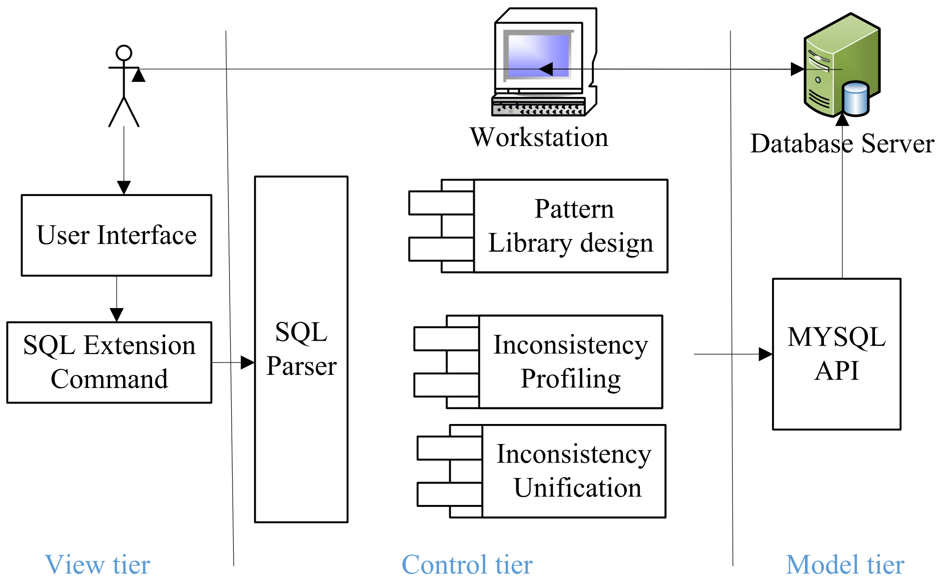


FIGURE 6.1: System architecture.

6.2 Subsystem Design

As shown in Figure 6.1, the system architecture is divided into three modules: SQL parser, inconsistency profiling, and inconsistency unification.

6.2.1 SQL Parser

The SQL Parser is implemented in C and the database is MYSQL. As is shown in Figure 6.2, Flex and Bison is employed to generate a C parser which could be compiled by the C compiler. Using these generated C codes, we implement the part of syntax check based on the MYSQL database and generate the grammar tree of the SQL extension query.

6.2.2 Inconsistency Profiling

This module aims to detect all possible patterns for each record and to provide a profiling result for users. Figure 6.3 shows the IPO (InputProcessOutput) diagram for this module. A modified DFA is used to execute the pattern detection process. There are three sub-modules according to the algorithms in Chapter 5: the construction of the FSM and the modified DFA, minimisation of the modified DFA, and the pattern detection process.

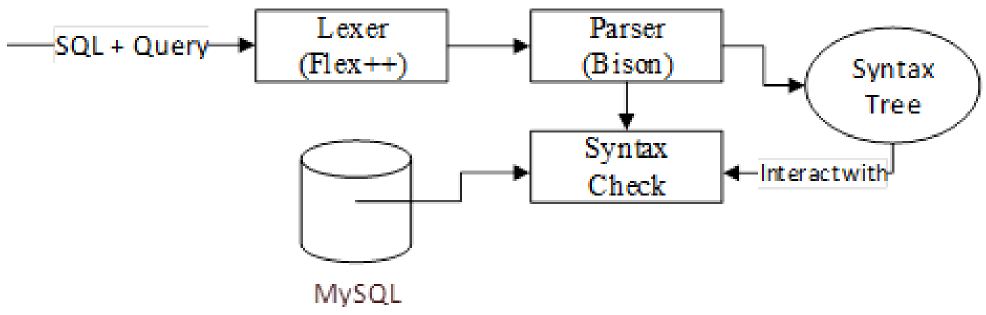


FIGURE 6.2: SQL parser.

- Construction of FSM and the modified DFA – the FSM is constructed using parallel and serial connections and it is transformed to a modified DFA using the modified subset construction algorithm as noted in Chapter 5.
- Minimisation of the modified DFA – the modified DFA is minimised without combining the separator states and states of different patterns.
- The pattern detection process – the DFA is processed with backtracking in the branch state to detect all possible patterns, and the pattern state label is used to split fields for each record.

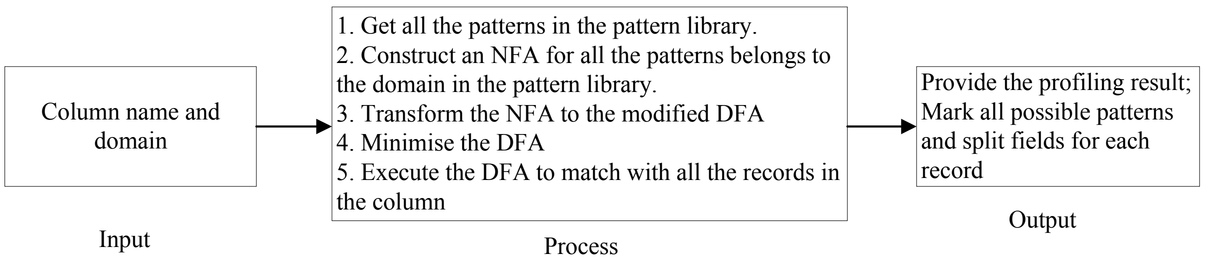

Column name and domain
Input
1. Get all the patterns in the pattern library.
2. Construct an NFA for all the patterns belongs to the domain in the pattern library.
3. Transform the NFA to the modified DFA
4. Minimise the DFA
5. Execute the DFA to match with all the records in the column
Process
Provide the profiling result; Mark all possible patterns and split fields for each record
Output

FIGURE 6.3: The IPO of inconsistency profiling module.

6.2.3 Inconsistency Unification

The inconsistency unification module transfers the target data records with different format patterns to a target format pattern. The IPO of the format transformation is shown in Figure 6.4. As mentioned above, the fields for each record are marked and split in the pattern detection process.

Based on the profiling results, the split substrings are later merged based on the structure of the target pattern.

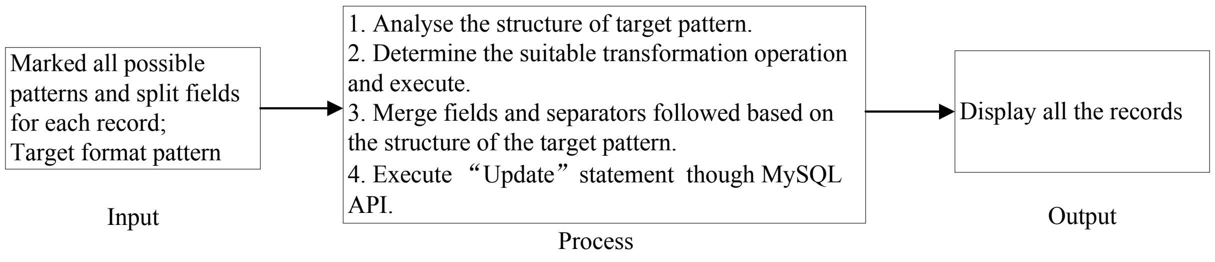

Marked all possible patterns and split fields for each record;
Target format pattern
Input
1. Analyse the structure of target pattern.
2. Determine the suitable transformation operation and execute.
3. Merge fields and separators followed based on the structure of the target pattern.
4. Execute "Update" statement though MySQL API.
Process
Display all the records
Output

FIGURE 6.4: The IPO of the inconsistency unification module.

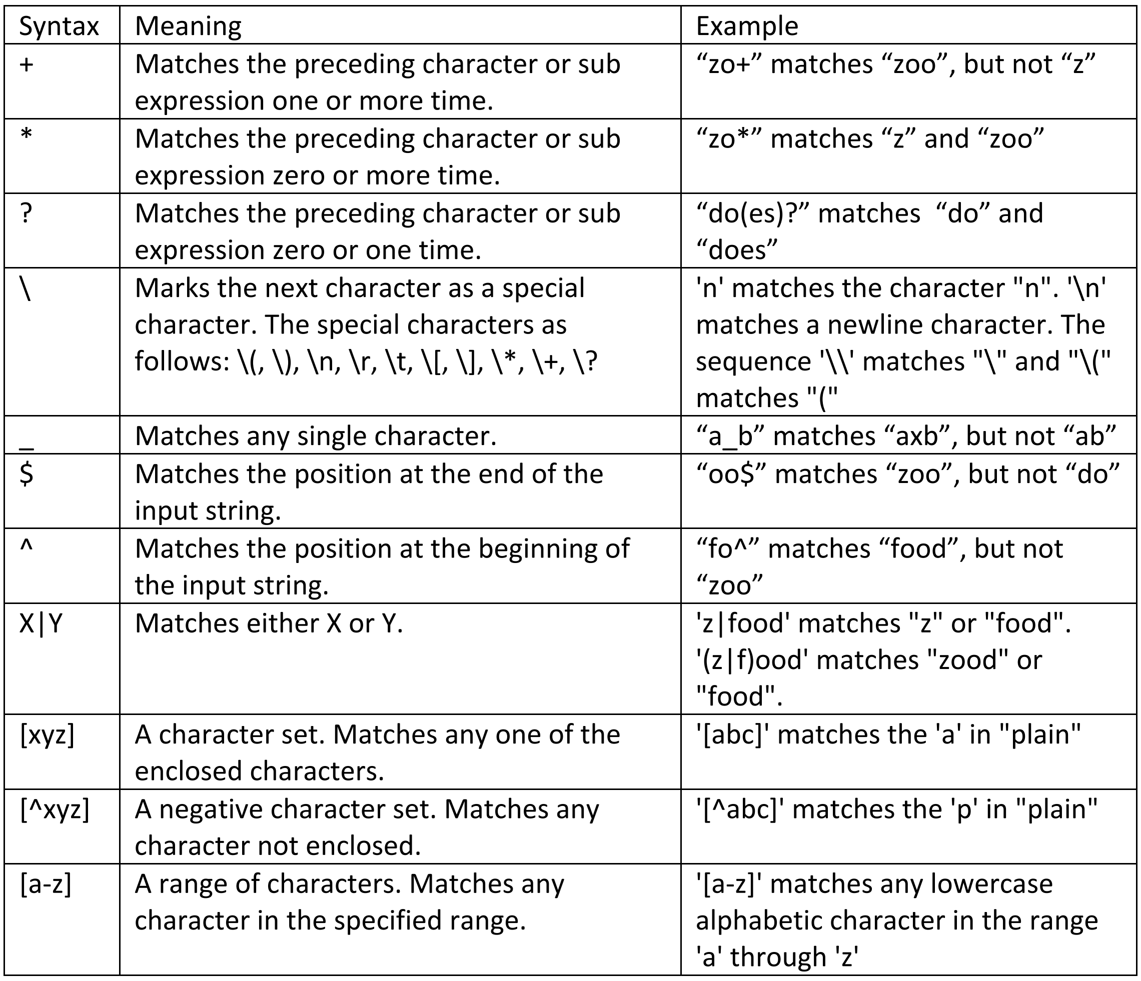
6.3 Limitations of regular expression patterns

According to Chapter 3, the format patterns were finally represented by regular expressions. Figure 6.5 lists the features of the regular expression patterns used in our inconsistency detection engine. The toolkit will not execute correctly when the regular expressions in the SQL extensions exceed the scope.

6.4 Use Case

In this section, we snapshot records in the gocard dataset and deal with the "Boardingstop" domain to show the interface and the usage of the toolkit. We use five steps with a snapshot of the toolkit to illustrate the usage.

- Step 1: Connect to a target database (Figure 6.6).
- Step 2: Find out the target column (In "BoardingStop" column in Figure 6.7, there are different format patterns such as "SD - Platform SN [ID]" and "SD (St SN) [ID]) and set the pattern library (Figure 6.8) for the domain of the column. If the target domain exists, the user could skip this step.
- Step 3: Profile the data with the pattern library. As shown in Figure 6.9, the profiling result includes a summary of the matched patterns, unmatched patterns and conflict patterns. The

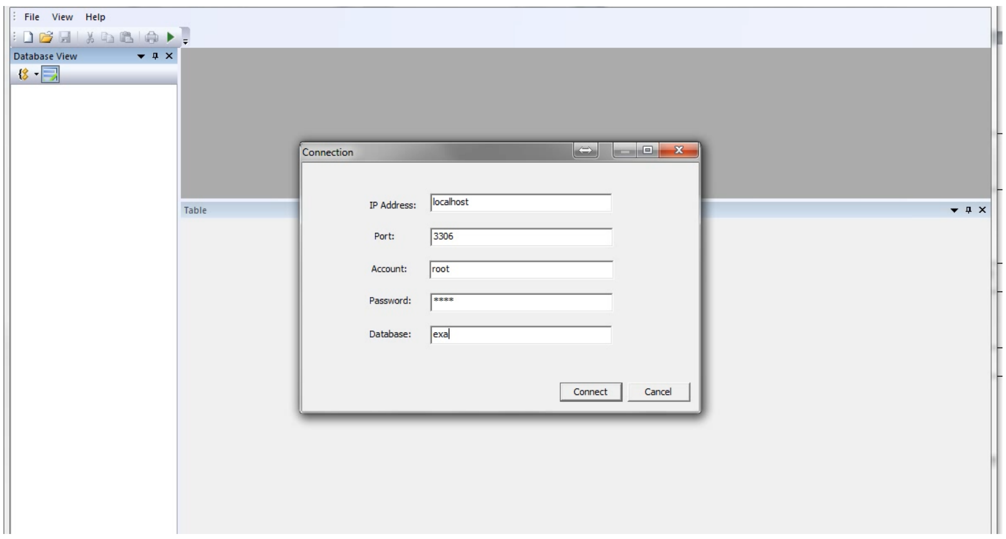


Syntax	Meaning	Example
+	Matches the preceding character or sub expression one or more time.	“zo+” matches “zoo”, but not “z”
*	Matches the preceding character or sub expression zero or more time.	“zo*” matches “z” and “zoo”
?	Matches the preceding character or sub expression zero or one time.	“do(es)?” matches “do” and “does”
\	Marks the next character as a special character. The special characters as follows: \(, \), \n, \r, \t, \[, \], *, \+, \?	'n' matches the character "n". '\n' matches a newline character. The sequence '\\' matches "\" and "\(" matches "("
_	Matches any single character.	“a_b” matches “axb”, but not “ab”
$	Matches the position at the end of the input string.	“oo$” matches “zoo”, but not “do”
^	Matches the position at the beginning of the input string.	“fo^” matches “food”, but not “zoo”
X|Y	Matches either X or Y.	'z|food' matches "z" or "food". '(z|f)ood' matches "zood" or "food".
[xyz]	A character set. Matches any one of the enclosed characters.	'[abc]' matches the 'a' in "plain"
[^xyz]	A negative character set. Matches any character not enclosed.	'[^abc]' matches the 'p' in "plain"
[a-z]	A range of characters. Matches any character in the specified range.	'[a-z]' matches any lowercase alphabetic character in the range 'a' through 'z'

FIGURE 6.5: Features of regular expressions in consistency detection engine.

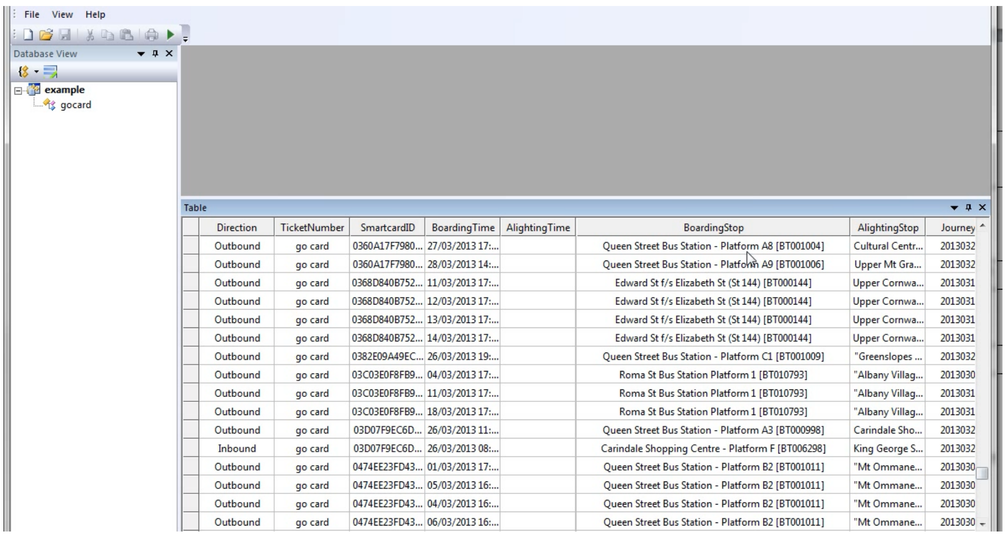
user could also check the details about the uncertain data to be guided to modify the pattern library. If the user needs to edit the pattern library based on the profiling result, it will back to trigger step 2.

- Step 4: Unify the inconsistent data. After the user verifies the profiling result, this step could be executed. The unification result is shown in Figure 6.10. Compared with the original records in Figure 6.7, all the records are represented by pattern “SD - SN [ID]”.



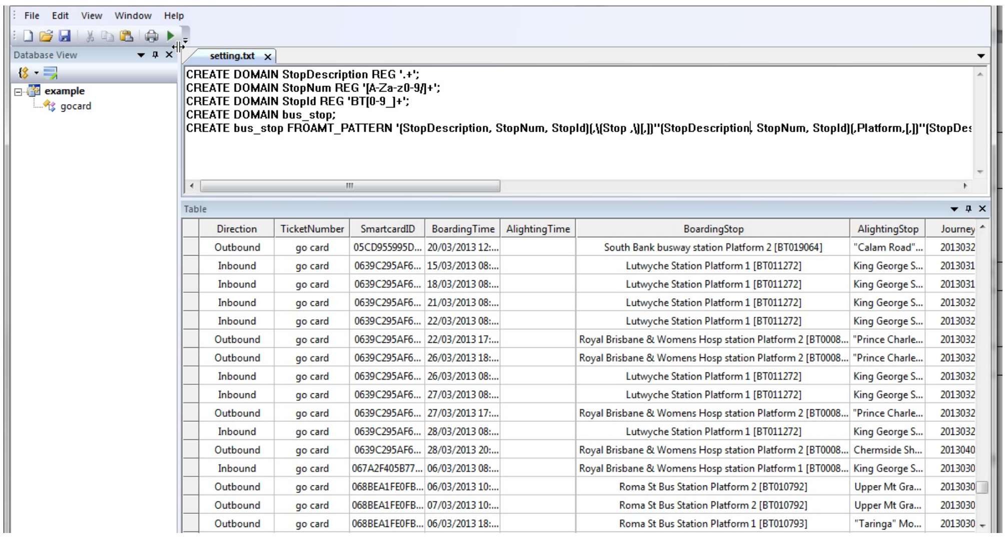
IP Address: localhost
Port: 3306
Account: root
Password: ****
Database: exa

FIGURE 6.6: Connection to a target database.



Direction	TicketNumber	SmartcardID	BoardingTime	AlightingTime	BoardingStop	AlightingStop	Journey
Outbound	go card	0360A17F7980...	27/03/2013 17:...		Queen Street Bus Station - Platform A8 [BT001004]	Cultural Centr...	2013032
Outbound	go card	0360A17F7980...	28/03/2013 14:...		Queen Street Bus Station - Platform A9 [BT001006]	Upper Mt Gra...	2013032
Outbound	go card	0368D840B752...	11/03/2013 17:...		Edward St f/s Elizabeth St (St 144) [BT000144]	Upper Cornwa...	2013031
Outbound	go card	0368D840B752...	12/03/2013 17:...		Edward St f/s Elizabeth St (St 144) [BT000144]	Upper Cornwa...	2013031
Outbound	go card	0368D840B752...	13/03/2013 17:...		Edward St f/s Elizabeth St (St 144) [BT000144]	Upper Cornwa...	2013031
Outbound	go card	0368D840B752...	14/03/2013 17:...		Edward St f/s Elizabeth St (St 144) [BT000144]	Upper Cornwa...	2013031
Outbound	go card	0382E09A49EC...	26/03/2013 19:...		Queen Street Bus Station - Platform C1 [BT001009]	"Greenslopes ...	2013032
Outbound	go card	03C03E0F8FB9...	04/03/2013 17:...		Roma St Bus Station Platform 1 [BT010793]	"Albany Villag...	2013030
Outbound	go card	03C03E0F8FB9...	11/03/2013 17:...		Roma St Bus Station Platform 1 [BT010793]	"Albany Villag...	2013031
Outbound	go card	03C03E0F8FB9...	18/03/2013 17:...		Roma St Bus Station Platform 1 [BT010793]	"Albany Villag...	2013031
Outbound	go card	03D07F9EC6D...	26/03/2013 11:...		Queen Street Bus Station - Platform A3 [BT000998]	Carindale Sho...	2013032
Inbound	go card	03D07F9EC6D...	26/03/2013 08:...		Carindale Shopping Centre - Platform F [BT006298]	King George S...	2013032
Outbound	go card	0474EE23FD43...	01/03/2013 17:...		Queen Street Bus Station - Platform B2 [BT001011]	"Mt Ommane...	2013030
Outbound	go card	0474EE23FD43...	05/03/2013 16:...		Queen Street Bus Station - Platform B2 [BT001011]	"Mt Ommane...	2013030
Outbound	go card	0474EE23FD43...	04/03/2013 16:...		Queen Street Bus Station - Platform B2 [BT001011]	"Mt Ommane...	2013030
Outbound	go card	0474EE23FD43...	06/03/2013 16:...		Queen Street Bus Station - Platform B2 [BT001011]	"Mt Ommane...	2013030

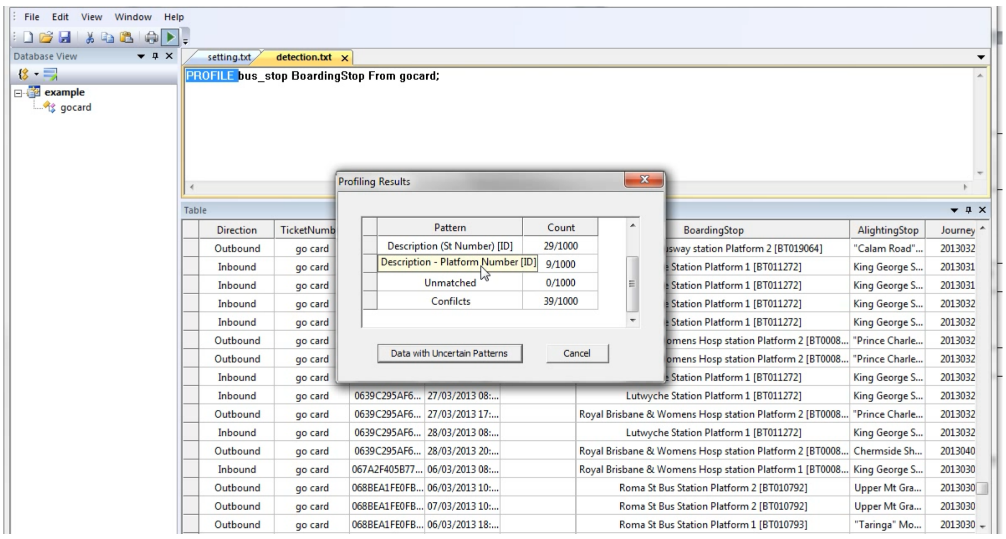
FIGURE 6.7: Snapshot of “BoardingStop” in the gocard data.



```
CREATE DOMAIN StopDescription REG '.+';
CREATE DOMAIN StopNum REG '[A-Za-z0-9/]+';
CREATE DOMAIN StopId REG 'BT[0-9_]+';
CREATE DOMAIN bus_stop;
CREATE bus_stop FROAMT_PATTERN '(StopDescription, StopNum, StopId)(,\(Stop ,\)[,])''(StopDescription, StopNum, StopId)(,Platform,[,])''(StopDes
```

Direction	TicketNumber	SmartcardID	BoardingTime	AlightingTime	BoardingStop	AlightingStop	Journey
Outbound	go card	05CD955995D...	20/03/2013 12:...		South Bank busway station Platform 2 [BT019064]	"Calam Road"...	2013032
Inbound	go card	0639C295AF6...	15/03/2013 08:...		Lutwyche Station Platform 1 [BT011272]	King George S...	2013031
Inbound	go card	0639C295AF6...	18/03/2013 08:...		Lutwyche Station Platform 1 [BT011272]	King George S...	2013031
Inbound	go card	0639C295AF6...	21/03/2013 08:...		Lutwyche Station Platform 1 [BT011272]	King George S...	2013032
Inbound	go card	0639C295AF6...	22/03/2013 08:...		Lutwyche Station Platform 1 [BT011272]	King George S...	2013032
Outbound	go card	0639C295AF6...	22/03/2013 17:...		Royal Brisbane & Womens Hosp station Platform 2 [BT0008...	"Prince Charle...	2013032
Outbound	go card	0639C295AF6...	26/03/2013 18:...		Royal Brisbane & Womens Hosp station Platform 2 [BT0008...	"Prince Charle...	2013032
Inbound	go card	0639C295AF6...	26/03/2013 08:...		Lutwyche Station Platform 1 [BT011272]	King George S...	2013032
Inbound	go card	0639C295AF6...	27/03/2013 08:...		Lutwyche Station Platform 1 [BT011272]	King George S...	2013032
Outbound	go card	0639C295AF6...	27/03/2013 17:...		Royal Brisbane & Womens Hosp station Platform 2 [BT0008...	"Prince Charle...	2013032
Inbound	go card	0639C295AF6...	28/03/2013 08:...		Lutwyche Station Platform 1 [BT011272]	King George S...	2013032
Outbound	go card	0639C295AF6...	28/03/2013 20:...		Royal Brisbane & Womens Hosp station Platform 2 [BT0008...	Chermside Sh...	2013040
Inbound	go card	067A2F405B77...	06/03/2013 08:...		Royal Brisbane & Womens Hosp station Platform 1 [BT0008...	King George S...	2013030
Outbound	go card	068BEA1FE0FB...	06/03/2013 10:...		Roma St Bus Station Platform 2 [BT010792]	Upper Mt Gra...	2013030
Outbound	go card	068BEA1FE0FB...	07/03/2013 10:...		Roma St Bus Station Platform 2 [BT010792]	Upper Mt Gra...	2013030
Outbound	go card	068BEA1FE0FB...	06/03/2013 18:...		Roma St Bus Station Platform 1 [BT010793]	"Taringa" Mo...	2013030

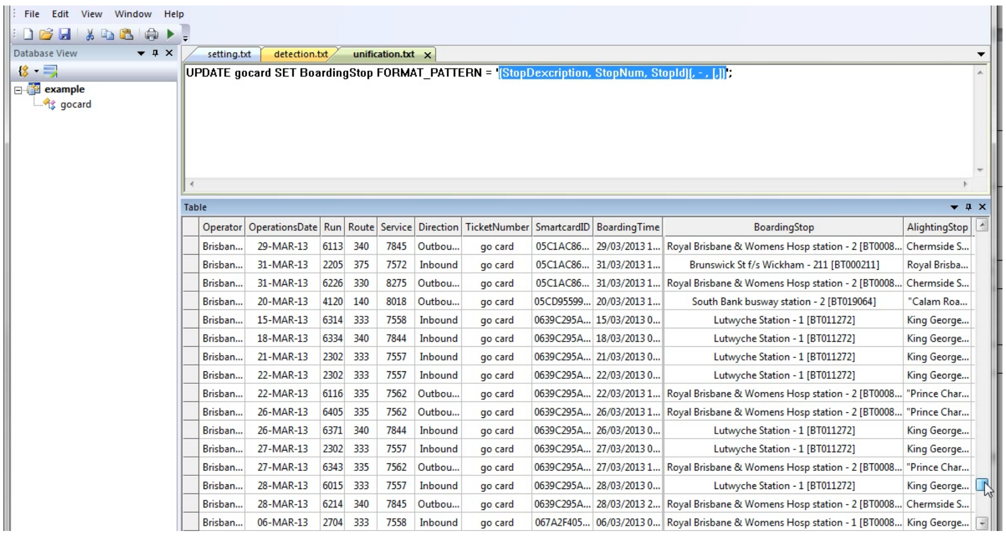
FIGURE 6.8: The pattern library setting.



```
PROFILE bus_stop BoardingStop From gocard;
```

Pattern	Count
Description (St Number) [ID]	29/1000
Description - Platform Number [ID]	9/1000
Unmatched	0/1000
Confilcts	39/1000

FIGURE 6.9: Profiling result.



```
UPDATE gocard SET BoardingStop FORMAT_PATTERN = '[StopDexcription, StopNum, StopId][, - , [,]]';
```

Operator	OperationsDate	Run	Route	Service	Direction	TicketNumber	SmartcardID	BoardingTime	BoardingStop	AlightingStop
Brisban...	29-MAR-13	6113	340	7845	Outbou...	go card	05C1AC86...	29/03/2013 1...	Royal Brisbane & Womens Hosp station - 2 [BT0008...	Chermside S...
Brisban...	31-MAR-13	2205	375	7572	Inbound	go card	05C1AC86...	31/03/2013 1...	Brunswick St f/s Wickham - 211 [BT000211]	Royal Brisba...
Brisban...	31-MAR-13	6226	330	8275	Outbou...	go card	05C1AC86...	31/03/2013 1...	Royal Brisbane & Womens Hosp station - 2 [BT0008...	Chermside S...
Brisban...	20-MAR-13	4120	140	8018	Outbou...	go card	05CD95599...	20/03/2013 1...	South Bank busway station - 2 [BT019064]	"Calam Roa...
Brisban...	15-MAR-13	6314	333	7558	Inbound	go card	0639C295A...	15/03/2013 0...	Lutwyche Station - 1 [BT011272]	King George...
Brisban...	18-MAR-13	6334	340	7844	Inbound	go card	0639C295A...	18/03/2013 0...	Lutwyche Station - 1 [BT011272]	King George...
Brisban...	21-MAR-13	2302	333	7557	Inbound	go card	0639C295A...	21/03/2013 0...	Lutwyche Station - 1 [BT011272]	King George...
Brisban...	22-MAR-13	2302	333	7557	Inbound	go card	0639C295A...	22/03/2013 0...	Lutwyche Station - 1 [BT011272]	King George...
Brisban...	22-MAR-13	6116	335	7562	Outbou...	go card	0639C295A...	22/03/2013 1...	Royal Brisbane & Womens Hosp station - 2 [BT0008...	"Prince Char...
Brisban...	26-MAR-13	6405	335	7562	Outbou...	go card	0639C295A...	26/03/2013 1...	Royal Brisbane & Womens Hosp station - 2 [BT0008...	"Prince Char...
Brisban...	26-MAR-13	6371	340	7844	Inbound	go card	0639C295A...	26/03/2013 0...	Lutwyche Station - 1 [BT011272]	King George...
Brisban...	27-MAR-13	2302	333	7557	Inbound	go card	0639C295A...	27/03/2013 0...	Lutwyche Station - 1 [BT011272]	King George...
Brisban...	27-MAR-13	6343	335	7562	Outbou...	go card	0639C295A...	27/03/2013 1...	Royal Brisbane & Womens Hosp station - 2 [BT0008...	"Prince Char...
Brisban...	28-MAR-13	6015	333	7557	Inbound	go card	0639C295A...	28/03/2013 0...	Lutwyche Station - 1 [BT011272]	King George...
Brisban...	28-MAR-13	6214	340	7845	Outbou...	go card	0639C295A...	28/03/2013 2...	Royal Brisbane & Womens Hosp station - 2 [BT0008...	Chermside S...
Brisban...	06-MAR-13	2704	333	7558	Inbound	go card	067A2F405...	06/03/2013 0...	Royal Brisbane & Womens Hosp station - 1 [BT0008...	King George...

FIGURE 6.10: Unification result.

Chapter 7

Conclusions and Future work

7.1 Summary of Contributions

In this thesis, we develop a user-driven pattern-based framework (see in Chapter 3) for domain agnostic data representational inconsistency problems and implement a toolkit (see in Chapter 6) to improve string data format consistency of data in a database. To complete this framework, we first design an SQL extension to describe the operator and rules based on the pattern features in Chapter 4. Then, before performing pattern unification in the datasets, we need to detect all possible patterns for each record in the target column of the dataset. Chapter 5 presents a modified DFA to deal with the pattern detection problem and experiments are designed to verify the accuracy and efficiency of the method. In Chapter 6, we introduce the technology applied in the implementation of our toolkit system via a use case.

More specifically, the contribution of this thesis are summarized as follows:

- We design an SQL extension based on the pattern features in Chapter 3 for the proposed framework, instead of writing complicated code specific to different format problems. The extensions were described in relation to each module of the pattern based framework, that is pattern design, inconsistency detection, and pattern unification (see in Chapter 4).

- We propose a pattern based framework with a complete and nearly mutually exclusive pattern library in the virtuous cycle of iterative and interactive approach in Chapter 3. Through this approach users are guided to complete the pattern library based on the profiling result.

A more complete pattern library leads to a better profiling result. This virtuous cycle reduces the amount of pre-knowledge users need to obtain.

- We study the problem of multiple regular expression rules matching avoid pairwise checking, which is in the module of pattern detection in the framework (see in 5). As discussed in 1, it is not a normal string matching problem. Therefore, we use regular expressions to represent the field features and separators to represent the structure and order. Since the final feature is represented by regular expressions with features, we develop a modified DFA with additional feature information. This method has proven to be more accurate and efficient compared with the naive method in the real world experiment.

- We propose a two-level pattern definition to combine both domain knowledge and regular expressions, and propose a split-transform-merge method to facilitate pattern unification (see in Chapter 3)

- We illustrate the technology employed in our toolkit in Chapter 6, and utilise a use case to show how the toolkit is used.

7.2 Directions for Future Work

In this section, we propose two promising future directions for our work.

7.2.1 Generate the Regular Patterns Automatically

In Chapter 5, we present an iterative and interactive method to collect the features of patterns from the users. Assuming that some users may not have extensive knowledge of regular expressions or the dataset, we expect one area of further research would be to generate the regular patterns automatically. It would also be of interest to investigate a method to summarize the different patterns in a string set and ensure completeness and accuracy. In addition, we also expect more studies of the expression of regular patterns. Theoretically, the patterns should be mutually exclusive. In real applications, this could never be achieved, however, we could reduce the coverage ratio between each pattern.

7.2.2 Improve the Method of Pattern Unification

In Chapter 3, we present a basic method (splittransformmerge method) to transform string data from one pattern to another. However, it still shows a weakness in relation to processing speed. We expect to employ the modified DFA to generate a path to convert strings. If we could modify each character in the string at the same time as performing the matching, the unification should be more efficient.

References

[1] A. V. Aho. *Compilers: Principles, Techniques and Tools (for Anna University), 2/e*. Pearson Education India, 2003.

[2] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[3] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 2006.

[4] V. Alfred. Algorithms for finding patterns in strings. *Algorithms and Complexity*, 1, 2014.

[5] J.-i. Aoe. *Computer algorithms: string pattern matching strategies*, volume 55. John Wiley & Sons, 1994.

[6] R. A. Baeza-Yates, C. Choffrut, and G. H. Gonnet. On Boyer-Moore automata. *Algorithmica*, 12(4-5):268–292, 1994.

[7] D. P. Ballou and H. L. Pazer. Modeling data and process quality in multi-input, multi-output information systems. *Management science*, 31(2):150–162, 1985.

[8] D. P. Ballou and H. L. Pazer. Modeling completeness versus consistency tradeoffs in information decision contexts. *Knowledge and Data Engineering, IEEE Transactions on*, 15(1):240–243, 2003.

[9] J. Barateiro and H. Galhardas. A survey of data quality tools. *Datenbank-Spektrum*, 14(15-21):48, 2005.

[10] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM Computing Surveys (CSUR)*, 41(3):16, 2009.

[11] C. Batini and M. Scannapieco. *Data quality: concepts, methodologies and techniques*. Springer, 2006.

[12] L. Bertossi and L. Bravo. Consistent query answers in virtual data integration systems. In *Inconsistency Tolerance*, pages 42–83. Springer, 2005.

[13] R. H. Blake and P. Mangiameli. Evaluating the semantic and representational consistency of interconnected structured and unstructured data. *AMCIS 2009 Proceedings*, page 126, 2009.

[14] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[15] F. Caruso, M. Cochinwala, U. Ganapathy, G. Lalk, and P. Missier. Telcordia's database reconciliation and data quality analysis tool. In *VLDB*, pages 615–618. Citeseer, 2000.

[16] S. Ceri, R. Cochrane, and J. Widom. Practical applications of triggers and constraints: Successes and lingering issues. In *VLDB*, pages 10–14, 2000.

[17] W. Chen, W. Fan, and S. Ma. Incorporating cardinality constraints and synonym rules into conditional functional dependencies. *Information Processing Letters*, 109(14):783–789, 2009.

[18] T. Churches, P. Christen, K. Lim, and J. X. Zhu. Preparation of name and address data for record linkage using hidden markov models. *BMC Medical Informatics and Decision Making*, 2(1):9, 2002.

[19] C. J. Coit, S. Staniford, and J. McAlerney. Towards faster string matching for intrusion detection or exceeding the speed of snort. In *DARPA Information Survivability Conference &amp; Exposition II, 2001. DISCEX'01. Proceedings*, volume 1, pages 367–373. IEEE, 2001.

[20] B. Commentz-Walter. *A string matching algorithm fast on the average*. Springer, 1979.

[21] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16, 2007.

[22] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Transactions on Database Systems (TODS)*, 33(2):6, 2008.

[23] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *In VLDB*, 3(1-2):173–184, 2010.

[24] J. E. Friedl. *Mastering regular expressions*. O'Reilly Media, Inc., 2002.

[25] H. Galhardas, D. Florescu, D. Shasha, E. Simon, C. Saita, et al. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.

[26] V. M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16(5):1, 1961.

[27] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *In VLDB*, 1(1):376–390, 2008.

[28] P. Gomes, J. Farinha, and M. J. Trigueiros. A data quality metamodel extension to CWM. In *Proceedings of the fourth Asia-Pacific conference on Conceptual Modelling-Volume 67*, pages 17–26. Australian Computer Society, Inc., 2007.

[29] J. M. Juran and A. B. Godfrey. Quality handbook. *Republished McGraw-Hill*, 1999.

[30] D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[31] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 339–350, New York, NY, USA, 2006. ACM.

[32] J. Kytöjoki, L. Salmela, and J. Tarhio. Tuning string matching for huge pattern sets. In *Combinatorial Pattern Matching*, pages 211–224. Springer, 2003.

[33] Y. W. Lee, L. L. Pipino, J. D. Funk, and R. Y. Wang. *Journey to Data Quality*. The MIT Press, 2006.

[34] M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

[35] Y. Li, X. Luo, X. Shao, and D. Wei. A regular expressions matching algorithm based on templates finite automata. In *Information and Communication Technology Convergence (ICTC), 2015 International Conference on*, pages 1058–1063. IEEE, 2015.

[36] A. Motro. Multiplex: a formal model for multidatabases and its implementation. In *International Workshop on Next Generation Information Technologies and Systems*, pages 138–158. Springer, 1999.

[37] A. Motro and P. Anokhin. Fusionplex: resolution of data inconsistencies in the integration of heterogeneous information sources. *Information Fusion*, 7(2):176–196, 2006.

[38] G. Myers. A Four Russians algorithm for regular expression pattern matching. *Journal of the ACM (JACM)*, 39(2):432–448, 1992.

[39] G. Navarro and M. Raffinot. *Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002.

[40] P. Oliveira, F. Rodrigues, P. Henriques, and H. Galhardas. A taxonomy of data quality problems. In *Second International Workshop on Data and Information Quality IQIS*. Citeseer, 2005.

[41] P. Oliveira, F. Rodrigues, and P. R. Henriques. A formal definition of data quality problems. In *IQ*, 2005.

[42] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[43] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[44] S. Sadiq. *Handbook of Data Quality Research and Practice*. Springer Verlag Berlin Heidelberg, 2013.

[45] I. Simon. *String matching algorithms and automata*. Springer, 1994.

[46] G. A. Stephen. *String searching algorithms*, volume 3. World Scientific, 1994.

[47] B. Stvilia, L. Gasser, M. B. Twidale, and L. C. Smith. A framework for information quality assessment. *Journal of the American Society for Information Science and Technology*, 58(12):1720–1733, 2007.

[48] K. Thompson. Programming techniques: Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, June 1968.

[49] C. Türker and M. Gertz. Semantic integrity support in sql: 1999 and commercial (object-) relational database management systems. *In VLDB*, 10(4):241–269, 2001.

[50] J. Wang and N. Tang. Towards dependable data repairing with fixing rules. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 457–468. ACM, 2014.

[51] R. Y. Wang. A product perspective on total management. *Communications of the ACM*, 41(2):58–65, 1998.

[52] R. Y. Wang and D. M. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems*, 12(4):5–33, 1996.

[53] F. Yu, Z. Chen, Y. Diao, T. Lakshman, and R. H. Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 93–102. ACM, 2006.

[54] Z. Yuan, B. Yang, X. Ren, and Y. Xue. Tfd: a multi-pattern matching algorithm for large-scale url filtering. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*, pages 359–363. IEEE, 2013.

[55] Z. Zhou, Y. Xue, J. Liu, W. Zhang, and J. Li. MDH: A high speed multi-phase dynamic hash string matching algorithm for large-scale pattern set. In *Information and Communications Security*, pages 201–215. Springer, 2007.

[56] J. Zobel and P. Dart. Phonetic string matching: Lessons from information retrieval. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 166–172. ACM, 1996.