A model-based Evaluation of Data Quality Activities in KDD☆

Mario Mezzanzanica[a,b], Roberto Boselli[a,b], Mirko Cesarini[a,b,*], Fabio Mercorio[b]

[a]*Department of Statistics and Quantitative Methods - University of Milano Bicocca, Milan, Italy*
[b]*CRISP Research Centre - University of Milano Bicocca, Milan, Italy*

Abstract

We live in the Information Age, where most of the personal, business, and administrative data are collected and managed electronically. However, poor data quality may affect the effectiveness of knowledge discovery processes, thus making the development of the data improvement steps a significant concern.

In this paper we propose the Multidimensional Robust Data Quality Analysis, a domain-independent technique aimed to improve data quality by evaluating the effectiveness of a black-box cleansing function. Here, the proposed approach has been realised through model checking techniques and then applied on a weakly structured dataset describing the working careers of millions of people. Our experimental outcomes show the effectiveness of our model-based approach for data quality as they provide a fine-grained analysis of both the source dataset and the cleansing procedures, enabling domain experts to identify the most relevant quality issues as well as the action points for improving the cleansing activities.

Finally, an anonymized version of the dataset and the analysis results have been made publicly available to the community.

Keywords: Data Quality; Data Cleansing; Model Checking; Real-life Application

1. Introduction

Nowadays, huge masses of people's data are available, thanks to the wide use of Information Systems, which represent the back-end of an increasing number of services and applications. Actually, public and private organizations recognise the value of data as a key asset to deeply understand social, economic, and business phenomena and to improve competitiveness in a dynamic business environment, as pointed out in several works (Fox et al., 1994; Madnick et al., 2009; Batini et al., 2009). Indeed, as Fayyad et al. (1996) remarks while introducing the KDD process, "the value of storing volumes of data depends on our ability to extract useful reports, events and trends, support decisions and policy based on statistical analysis and inference." In the last years, the data quality improvement and analysis techniques have become an essential part of the KDD process as they contribute to guarantee the believability of the overall knowledge process[1], making the reasoning over data a very significant concern (Sadiq, 2013; Fisher et al., 2012; Holzinger et al., 2013b; Pasi et al., 2013a; Herrera-Viedma & Peis, 2003). In this paper we aim to draw the attention to data quality in the context of KDD.

Indeed, most researchers agree that quality of data is frequently poor, and this represents a problem in practical applications of KDD since according to the "garbage in, garbage out" principle, dirty data can have unpredictable

☆This work is partially supported within a Research Project granted by the CRISP Research Centre (Interuniversity Research Centre on Public Services - `http://www.crisp-org.it`) and Arifl (Regional Agency for Education and Labour - `http://www.arifl.regione.lombardia.it`).

*Corresponding Author. Mirko Cesarini, Ph.D. Assistant Professor at University of Milano-Bicocca, Department of Statistics and Quantitative Methods. Via Bicocca degli Arcimboldi 8, II floor. I-20126 Milano, Italy. Tel (+39)02-6448-5849. Fax (+39)02-6448-5878. email: mirko.cesarini@unimib.it

[1]Here the term *believability* is intended as "the extent to which data are accepted or regarded as true, real and credible"(Wang & Strong, 1996)

effects on the information derived from them, as noted by Fox et al. (1994); Levitin & Redman (1995); Ballou & Tayi (1999); Hipp et al. (2001); Haug et al. (2011); Dasu (2013).

In recent years industrial and academic communities have spent a great effort to address data quality issues (e.g., by performing quality analysis and improvement, data visualisation and management, data cleansing, etc.) both from a practical and a theoretical point of view, as studied by Barateiro & Galhardas (2005); Pipino et al. (2002); Wang & Strong (1996). In this regard, Batini & Scannapieco (2006) reported that a gap between practice-oriented approaches and formal research contributions still exists in this field. Indeed, from an industry perspective, a lot of off-the-shelf tools are available, but often they lack of formality in addressing domain independent problems, as the case of the ETL tools[2]. In such tools a quite relevant amount of the data quality analysis and cleansing design has still to be done manually or by ad-hoc developed routines, that may be difficult to write and maintain (Rahm & Do, 2000). On the other side, theoretical formalisms are sound and rigorous, but they often require a strong background from practitioners, reason that prevents their large-scale diffusion.

Within this work we support the idea that model-based verification approaches (model checking for instance) can support the Data Quality task of the KDD process in real-life situations by

(i) modelling data evolution over time in a natural way (e.g, as path on a graph). This allows domain experts to concentrate on *what* quality constraints need to be modelled rather than *how* to verify them, thus supporting the definition and formalisation of domain related quality requirements;

(ii) evaluating the effectiveness of cleansing activities performed through a practice-oriented approach (like the Extraction, Transformation, and Loading used in data warehousing).

In this regard, here we present the Multidimensional Robust Data Quality Analysis, a novel technique we defined to formalise and automatically verify both the quality of the data and the robustness of an industrial cleansing process. The technique has been realised by using a model-checking based tool. Furthermore, we report our experience in the application of such technique to a public administration dataset composed by more than 21 million items framed in the context of the Italian Labour Market Domain, then providing a (smaller) database and the experimental results to the community.

2. Motivation and Contribution

Huge amounts of data describing people behaviours are collected by the Information Systems of enterprises and organizations. Such data often have an unexpressed informative power, indeed the study of relations and correlations among them allows domain experts to understand the evolution of subtended behaviours or phenomena over time, as recently outlined by Holzinger (2012, 2011); Wong et al. (2011); Lovaglio & Mezzanzanica (2013). Among the time-related data, the *longitudinal data* (i.e., repeated observations of a given subject, object or phenomena at distinct time points, see, e.g.,Bartolucci et al. (2012)) have received much attention from several academic research communities as they are well-suited to model many real-world instances, including labour and healthcare domains, see, e.g. (Hansen & Järvelin, 2005; Holzinger, 2012; Holzinger & Zupan, 2013; Prinzie & Van den Poel, 2011; Lovaglio & Mezzanzanica, 2013; Devaraj & Kohli, 2000).

In such a context graphs or tree formalisms, which are exploited to model *weakly-structured* data, are deemed also appropriate to model the expected data behaviour, that formalise how the data should evolve over time. In this regard, Holzinger (2012) has recently clarified that a relationship exists between weakly-structured data and time-related data. Namely, let $Y(t)$ be an ordered sequence of observed data, e.g., subject data sampled at different time $t \in T$, the observed data $Y(t)$ are weakly structured if and only if the trajectory of $Y(t)$ resembles a random walk (on a graph). The following example should help in clarifying the matter.

[2]The ETL (Extract, Transform and Load) is an approach supporting the data preprocessing and transformation tasks in the KDD process (Fayyad et al., 1996). The data extracted from a source system undergo a set of transformations that analyse, manipulate and then cleanse the data before loading them into a Datawarehouse.

Table 1: An example of a Mobile Phone Tracking Dataset

Event-ID	Event Type	Cell-ID	Timestamp
01	Cell-IN	3902	12/01/2011:08::35:00
02	Traffic	3902	12/01/2011:11::00:05
03	Traffic	3902	12/01/2011:13::10:15
04	Traffic	3902	12/01/2011:18::45:55
05	Cell-IN	40122	12/01/2011:22::00:00
...	...	...	...

Motivating Example. Let us introduce the *Mobile Phone Tracking Example*. The dataset in Tab. 1 shows the events recorded by a mobile telephone operator for lawful interception purposes.[3] The data describe mobile phones connecting to cells of a cellular network, performing calls, exchanging messages, and data packets. Such data represent a log of the activities that a law enforcement agency can request for investigation. Each record reports information about: the MS-ID (Mobile Station ID, i.e. an ID identifying the mobile phone involved); the BTS-ID (the ID of the base transceiver station to which the Mobile Phone is connected); and the Event-Type. For the sake of simplicity, we reduce the several existing event types to *cell-in*, *cell-out*, and *traffic*. The *cell-in* event happens when a mobile phone starts being served by a BTS (Base Transceiver Station), e.g. the mobile phone is switched on or it enters into the BTS coverage area. The *cell-out* event takes place when the mobile phone is no longer served by the BTS where it has previously performed a *cell-in* (this can be due to the mobile phone being switched-off, or to the exit from the BTS coverage area). The *traffic* event is recorded when a call is initiated, or a message is sent or received, or some data are exchanged by the phone. The Timestamp value reports the call start time or the message/data packet send time.

Intuitively, one could model the longitudinal data evolution on a graph, then it could apply any graph-search to verify if the longitudinal data sequence (i.e., the trajectory) is "correct" or not (i.e., if it satisfies or not a set of quality requirements). To this aim, a mobile phone event sequence should evolve according to the automaton described in Fig. 1(a). Unfortunately, the real data do not fully comply with these criteria: several *cell-in* can be found in the same cell (with no *cell-out* in between), several traffic events have no previous *cell-in* on the BTS, etc. This is mainly due to signal drop issues affecting the radio connections. Let us suppose that the elapsed intervals should be computed for analysis purposes i.e., the intervals when a mobile phone is served by (and thus being into) a BTS. Unfortunately the data quality issue may prevent or affect such intervals computation. Note that, as we discuss in Sec. 3, modelling such quality requirements through functional dependencies (FDs) may be a hard task since they mainly work on attributes rather than tuples, even though their expressivity has been recently revisited and improved, see Bravo et al. (2008).

Actually, evaluating and improving the quality of a data source archive and, in turn, the effectiveness of a cleansing process is a challenging task while the comparison between archive contents against real data is often either unfeasible or very expensive (e.g. lack of alternative data sources, cost for collecting the real data, etc.). In such a case, cleansing procedures based on business rules still represent the most adopted solution by industry, as proved by the diffusion of several open source and commercial tools, see (Thomsen & Pedersen, 2005; Barateiro & Galhardas, 2005) for a survey. A reliable answer to questions like "How good are the adopted data cleansing processes?" becomes quite relevant, especially when formalising and measuring such "goodness" can strengthen the believability of the overall knowledge discovery process.

Here we support the idea that a model-driven verification of data cleansing activities can strengthen the effectiveness of the KDD process, by providing to data mining algorithms a more reliable cleansed dataset. The contribution of this paper, which extends preliminary results from (Boselli et al., 2013), goes into three directions.

- First, we present and formalise the Multidimensional Robust Data Quality Analysis (MRDQA for short), a domain-independent iterative technique aimed to evaluate the effectiveness of a black-box cleansing function over a dirty dataset. Then, a visualization technique is used to facilitate the understanding and assessment of the MRDQA results, namely the parallel-coordinates;

[3] Lawful Interception is a security process where a service provider or a network operator collects individuals intercepted data or communications on behalf of law enforcement officials, see (European Telecommunications Standards Institute ES 201 671, 2009) for more details.

- Second, we express the task of evaluating weakly structured data quality as a model checking problem, then we implemented the MRDQA using the UPMurphi tool (Della Penna et al., 2009);
- Third, we apply the MRDQA on a real-life government application in the field of Labour Market (The Italian Ministry of Labour and Welfare, 2012). Finally, a smaller version of the dataset we analysed and a demo are made available on line to the community.

The outline of this paper is as follows. In the next section we provide an overview of the related work while in Sec. 4 we introduce some background notions about data quality, model checking and the interaction between them. Then, in Sec. 5 we present the Multidimensional Robust Data Quality Analysis while in Sec. 6 we introduce the labour market domain. Sec. 7 extensively draws the experimental results as well as the characteristics of the online database. Finally, in Sec. 8 we sketch some concluding remarks providing in the appendix the code used to model the labour market domain.

3. Related Work

The data quality analysis and improvement tasks have been the focus of a large body of research in different domains, that involve statisticians, mathematicians and computer scientists, working in close cooperation with application domain experts, each one focusing on its own perspective (Abello et al., 2002; Fisher et al., 2012).

To give a few examples, statisticians always fought for better data quality by applying: data mining and machine learning techniques for data edits (Mayfield et al., 2010; Winkler, 1997; Fellegi & Holt, 1976), probabilistic record linkage (Winkler, 2000; Fellegi & Sunter, 1969; Newcombe & Kennedy, 1962), and error detection (Elmagarmid et al., 2007; Winkler, 2004). On the other side, computer scientists developed algorithms and tools to ensure data correctness by paying attention to the whole Knowledge Discovery process, from the collection or entry stage to data visualisation (Holzinger et al., 2013a; Ferreira de Oliveira & Levkowitz, 2003; Clemente et al., 2012; Fox et al., 1994), exploiting both hard and soft computing techniques, see e.g. (Bertossi, 2006; Chomicki & Marcinkowski, 2005b; Hipp et al., 2001; Yu et al., 2006).

Usually, the quality evaluation task in the literature is related to the data *cleansing* (or *cleaning*) problem, which basically consists in the identification of a set of activities to cleanse a dirty dataset. In this regard, a common technique is record linkage (also known as *object identification*, *record matching*, *merge-purge problem*) which aims to bring together corresponding records from two or more data sources. The purpose is to link the data to a corresponding higher quality version and to compare them (Elmagarmid et al., 2007). An alternative approach uses *Business Rules* identified by domain experts to cleanse the dirty data. The cleansing procedures can be implemented in SQL or in other tool specific languages.

This paper handles the problem of data quality verification in terms of *consistency* (as specified in Sec. 4.1) by mapping both the data dynamics and the consistency constraints over a finite state system, then using model checking to verify them.

Finite State Systems in the context of data (and Formal Methods in general) have been investigated in the areas of *databases* and *artificial intelligence*. Chomicki (1995) basically encodes bounded database history over Büchi automata to check temporal constraints. The purpose of Chomicki is to build an efficient framework to perform temporal queries on databases while no attention is paid to the data quality issues. Indeed, the author declares that the work focuses on transaction time databases and it is assumed that the stored data *exactly* correspond to the real world ones. Formal verification techniques were applied to databases with the aim to prove the termination of triggers by exploiting both *explicit* model checking (Choi et al., 2006) and *symbolic* techniques (Ray & Ray, 2001). The use of CTL model checking has been investigated for semistructured data retrieval, whether XML based (Neven, 2002) or web based (Dovier & Quintarelli, 2002) as well as to solve queries on semistructured data (Dovier & Quintarelli, 2009; Afanasiev et al., 2004; Dovier & Quintarelli, 2002).

In the database area, a lot of works have been focusing on *constraint-based data repair* for identifying errors by exploiting FDs (Functional Dependencies), multivalued dependencies, join dependencies, and inclusion dependencies. However, as introduced in Sec. 4.1, they are not suited for specifying constraints on longitudinal or historical

data (Vardi, 1987; Chomicki, 1995; Fan, 2008). Specifically, Vardi (1987) motivated the usefulness of formal systems in databases by observing that FDs are only a fragment of the first-order logic used in formal methods while Fan et al. (2010) observed that, even though FDs allow one to detect the presence of errors, they have a limited usefulness since they fall short of guiding one in correcting the errors.

Two very effective approaches based on FDs are *database repair* (Chomicki & Marcinkowski, 2005a; Greco et al., 2001) and *consistent query answering* (Arenas et al., 1999; Bertossi, 2006). The former aims to find a *repair*, namely a database instance that satisfies integrity constraints and minimally differs from the original (maybe inconsistent) one. The latter approach tries to compute *consistent query answers* in response to a query, namely answers that are true in every repair of the given database, but the source data is not fixed. Unfortunately, finding consistent answers to aggregate queries is a NP-complete problem already using two (or more) FDs (Bertossi, 2006; Chomicki & Marcinkowski, 2005b). To mitigate this problem, recently a number of works have exploited heuristics to find a database repair, as (Yakout et al., 2013; Cong et al., 2007; Kolahi & Lakshmanan, 2009). They seem to be very promising approaches, even though their effectiveness have not been evaluated on real-world domains.

More recently, the NADEEF (Dallachiesa et al., 2013) tool has been developed for creating a unified framework able to merge the most used cleansing solutions by both academy and industry. It provides a programming interface that would facilitate the user in expressing quality constraints - and thus in cleansing the data - through business rules, conditional functional dependencies, matching dependencies, and denial constraints. In our opinion NADEEF gives an important contribution in the field of data cleansing also providing an exhaustive overview about the most recent (and efficient) solutions for cleansing the data. Indeed, as the authors remark, consistency requirements are usually defined on either a single tuple, two tuples or a set of tuples. The first two classes are enough for covering a wide spectrum of basic data quality requirements for which FD-based approaches are well-suited. However, the latter class of quality constraints (that NADEEF does not take into account according to its authors) requires reasoning with a finite (but not bounded) set of data items over time as the case of longitudinal data, and this makes the exploration-based technique a good candidate for that task. More specifically, AI planning can enable domain experts to express complex quality requirements and to effortlessly identify best suited cleansing actions for a particular data quality context.

Finally, from an industry point of view, a lot of off-the-shelf tools are available and well-supported, but they often lack of formality in addressing domain independent problems, as the case of several ETL tools[4]. In such tools a quite relevant amount of the data analysis and cleaning work has still to be done manually or by ad-hoc routines, that may be difficult to write and maintain, as discussed by Rahm & Do (2000).

4. Background

In this section we first briefly discuss about data quality, then we introduce some background notions about the model checking technique and the UPMurphi tool, which will be used in the remainder of the paper. Finally, we connect all these concepts by introducing the finite state event databases and datasets, then describing how to perform data consistency verification through model checking.

4.1. Data Quality at a Glance

Data quality is a broad concept and it has been widely addressed in the literature of several research communities. In this regard, the most common and concise definition of data quality is given by Wang & Strong (1996), which define data quality as "fitness for use". This implies that data quality is a domain and goal dependent concept, thus a dataset can be considered appropriate for one use while may not be suitable for a different one. In such direction, Kahn et al. (2002) considers the quality of data as "the extent to which data are conform to a given specification" while, in a more extensive way, Redman (2001) states that "data are of high quality if they are fit for their intended uses in operations, decision making, and planning. Data are fit for use if they are free of defects and possess desired features".

In this paper we focus on *consistency*, a quality dimension which Batini & Scannapieco (2006) refer to "the violation of semantic rules defined over (a set of) data items where e.g., items can be tuples of relational tables or records in a file".

[4]In the ETL approach (Extract, Transform and Load) data extracted from a source system pass through a sequence of transformations, that analyse, manipulate and then cleanse the data before loading them into a Datawarehouse

4.2. Explicit Model Checking

Model checking (see e.g., Clarke et al. (1999); Baier et al. (2008)) is a hardware/software verification technique to determine whether a model system obeys a specification of its intended behaviour. The model is described in terms of *state variables*, whose evaluation determines a state, and *transition relations* between states, which specify how the system can move from a state to the next one as a consequence of a given input action.

Generally speaking, given a model of a dynamic system (i.e., a transition system) and a formal property to be verified on it, a model checking tool verifies whether this property holds for each state of that model or not. In the latter case, the model checker returns the *error-trace*, describing how the system reached the error.

Focusing on *explicit* model checking, it performs an exhaustive search on the transition system and it progressively collects the complete set of the system states (also called the system *state space*). Thus, such technique can work on *Finite State System (FSS for short)* only. To clarify this concept, let us consider a system model with a finite set of state variables $x_1, x_2, \ldots, x_n$. If each variable x_i ranges over a (nonempty) set D_i of values, the state space is $S = D_1 \times \cdots \times D_n$, which enumerates all the possible system behaviours. If a particular system state cannot be actually reached (e.g., a variable can never be set to a specific value, even if it is in its domain), it will be never generated or analysed. This kind of state space exploration (limitation) is also called *reachability analysis*.

Model checking algorithms are subject to the *state explosion problem*, since managing a large state space may require a very big storage space (usually RAM): however, the ability to generate only the system reachable states, and several space saving techniques helps mitigating the problem, see e.g. (Edelkamp & Jabbar, 2006; Della Penna et al., 2004).

For the sake of clarity, we formalise an FSS as follows.

Definition 1 (Finite State System)**.** *A* Finite State System *(*FSS*)* $\mathcal{S}$ *is a 4-tuple* (S,I,A,F)*, where:* S *is a finite set of* states*,* $I \subseteq S$ *is a finite set of* initial states*,* A *is a finite set of* actions *and* $F : S \times A \to S$ *is the* transition function*, i.e.* $F(s,a) = s'$ *iff the system from state* s *can reach state* s' *via action* a*. Hence, we define:*

- *a* trajectory *as a sequence* $\pi = s_0 a_0 s_1 a_1 s_2 a_2 \ldots a_{n-1} s_n$ *where,* $\forall j = 0, \ldots, n$*,* $s_j \in S$ *is a state,* $\forall i = 0, \ldots, n-1$*,* $a_i \in A$ *is an action and* $F(s_i, a_i) = s_{i+1}$*. If* π *is a trajectory, we denote with* $|\pi|$ *the length of* π *given by the number of actions; Finally, we write* $\pi_s(k)$ *(resp.* $\pi_a(k)$*) to denote the state* s_k *(resp. the action* a_k*);*
- Reach(S) *as the set of all states reachable from the initial ones.*

Let $\mathcal{S}$ be an FSS according to Def. 1 and let φ be a formula specifying a property to be satisfied on the system. Let a state $s_E \in E$ be an error state if the invariant formula φ is not satisfied. Then, we can define the set of *error states* $E \subseteq S$ as the union of the states violating φ. Moreover, we limit the error exploration to at most T actions (the finite horizon), i.e. only sequences reaching an error $s_E \in E$ within the finite horizon are detected. Note that this restriction has a limited practical impact in our contexts although being theoretically quite relevant, see Kroening et al. (2003); Biere et al. (2003).

Model checking is traditionally used to explore and verify all the feasible execution paths of a system. Then, informally speaking a *model checking problem* is composed by a description of the FSS to be explored, a property to verify and a finite horizon. A feasible solution, i.e. the error trace (if any), is a trajectory leading the system from an initial state to an error one. More formally we can define the following.

Definition 2 (Model Checking Problem and Solution)**.** *Let* $\mathcal{S} = (S, I, A, F)$ *be an FSS. Then, a* model checking problem *(*MCP *in the following) is a triple* $\mathcal{P} = (\mathcal{S}, \varphi, T)$ *where* φ *is the the invariant condition and* T *is the finite horizon.*
Then a feasible solution for $\mathcal{P}$ *is a reachable trajectory* π *in Reach*(S) *s.t.:* $\exists s_I \in I$*,* $|\pi| = k, k \leq T, \pi_s(1) = s_I$ *and* $\pi_s(k) \in E$*.*

4.2.1. The UPMurphi tool

The high formalisation and computational power of model checking has been applied to several contexts away from the HW/SW verification, from Fault-Tolerance to Control and AI Planning domains, see e.g. (Atlee & Gannon, 1993; Giunchiglia & Traverso, 2000). As a consequence, a number of model checking tools have been adapted and enhanced properly, see Bérard et al. (2010) for a survey. Here we use the UPMurphi tool (Della Penna et al.,

2012, 2009), a model-checking-based universal planner containing algorithms directly derived from the explicit model checker Murphi (Dill, 1996). A detailed description of Murphi is out of the scope of this paper, however it is important to highlight that Murphi naturally supports first-order logic quantifiers in the model specification, which are unravelled through state enumeration.

Note that, even though UPMurphi is a planner, it exploits the well-known planning-via-model-checking paradigm (Giunchiglia & Traverso, 2000), that allows one to use a model checker for searching the planning state space, stopping the search when a goal state is found.

Moreover, UPMurphi has proved its effectiveness in dealing with several AI planning problems in both deterministic and non-deterministic domains (Fox et al., 2012, 2011; Della Penna et al., 2009, 2010a, 2011, 2010b, 2012). Finally, UPMurphi presents three features useful for our purposes:

- it allows the use of C/C++ language constructs to model complex dynamics and to exploit external libraries to connect with other services (e.g., databases through ODBC drivers). This feature helps us to directly access to external data sources, since restrictions on accessing and duplicating archives are frequently enforced due to non-disclosure and secrecy agreements;
- it was enhanced with a disk-based algorithm (Mercorio, 2013) for exploring the system dynamics, thus enabling the visit of huge graphs.
- it is also a *Universal Planner*. A Universal Plan, first introduced by Schoppers (1987) is a set of policies, computed off-line, able to bring the system to the goal from any state reachable from the initial ones, the reader can see the contributions of Cimatti et al. (1998); Della Penna et al. (2012) for details. The universal planner output is a table (also known as controller) of <state,action> pairs describing which actions can be performed from each state to reach the goal. In this work we modify the UPMurphi universal plan algorithm to synthesise a *Universal Checker*, i.e., a taxonomy of all the inconsistencies affecting the data, as described in detail in Sec. 7.3.

4.3. Data Consistency Verification as Model Checking Problem

Finite State Systems have been widely applied in the literature, also dealing with event-driven systems by mapping events onto actions e.g., see (Holzmann & Smith, 1999; Atlee & Gannon, 1993). Similarly, an Information System recording longitudinal data in a database can be viewed as an event-driven system by considering a database record as an *event*, i.e. a record content or a subset thereof is interpreted as the description of an external world event modifying the system state, while an ordered set of records represents an *event sequence*.

To this aim we introduce the following.

Definition 3. *Let $\mathcal{R} = (R_1, \ldots, R_n)$ be a schema of a database relation.*

- *An* event $e = (r_1, \ldots, r_m)$ *is a* record *of the projection $(R_1, \ldots, R_m)$ over $\mathcal{Q} \subseteq \mathcal{R}$ with $m \leq n$, such that $r_1 \in R_1, \ldots, r_m \in R_m$;*
- *An* event sequence *is a $\sim$-ordered sequence of events $\epsilon = e_1, \ldots, e_n$. Indeed, a* total order *relation $\sim$ on events can be defined such that $e_1 \sim e_2 \sim \ldots \sim e_n$;*
- *A* Finite State Event Dataset *(FSED) S_i is an event sequence derived from a longitudinal dataset, while a* Finite State Event Database *(FSEDB) is a database S whose content is $S = \bigcup_{i=1}^{k} S_i$ where $k \geq 1$.*

In the following we denote by ϵ_i a subsequence of ϵ from the first event to the one in position i.

In several domains it is advisable to split a (large) dataset into different subsets (e.g., for improving scalability). Then, each subset can be managed separately (e.g., parallel computation can be performed). From now on, the term FSED will be used to refer to a subset while the overall dataset will be called FSEDB.

Intuitively, the application of model checking techniques to data quality problems (as introduced in (Mezzanzanica et al., 2011)) is driven by the idea that a *model* describing the expected evolution of any event sequence can be used to verify if a dataset (called *actual data*) retrieved from a data source (e.g., a database) is compliant with such model.

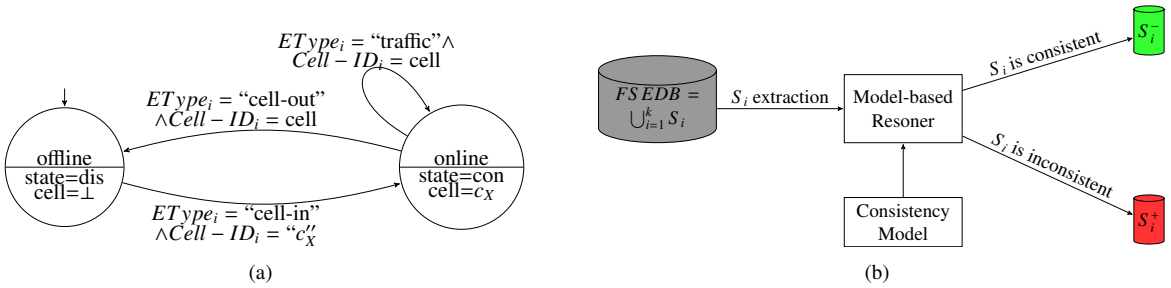


Figure 1: (a) A Graphical representation of the consistency model for the Mobile Phone Tracking domain where the lower part of a node describes how the system state evolves when an event happens. (b) A Graphical representation of a model-based data consistency verification on an FSEDB. Each S_i is extracted from the FSEDB and analyzed according to the Consistency Model provided, then each S_i is inserted either into the set of consistent ones (S_i^-) or into the set of inconsistent ones (S_i^+).

Then, the data consistency verification problem can be expressed as a model checking problem on FSSs: a solution for the latter (if any) is an inconsistent set of records for the former.

Performing a model-based data consistency evaluation requires the following steps: (1) to define a model of the data evolution, (2) to identify the consistency rules to be verified and (3) to verify the data source (e.g., the FSEDB introduced before) against the data evolution model and the consistency rules. A schematic representation of how this task can be accomplished by using a model checker is depicted in Fig. 1(b), which works as follows. From here on, without loss of generality, we refer to the consistency model as a model which encapsulates both the model of the data evolution and the consistency properties [5].

Step 1 (Domain Modelling) A domain expert defines the consistency model describing the correct evolution of the data through the model checking tool language.

Step 2 (Data Verification) An FSED S_i is retrieved from the FSEDB source S. The model checker looks for an error trace. A solution (if any) represents an inconsistency affecting the dataset S_i. Otherwise S_i is considered consistent.

Step 3 (Iteration) Repeat step 2 for each $S_i \in S$.

Working Example. The following example should clarify the matter. Let us consider the dataset introduced in Tab. 1. The information collected about a single mobile phone is an FSED S_i, while the information of several of them is the FSEDB S.

An *event* e_i is composed of the attributes *MS-ID*, *Event Type*, *Cell-ID*, and *Timestamp*, namely $e_i = (MS - ID_i, EType_i, Cell - ID_i, Timestamp_i)$. Moreover, the total-order operator $\sim$ could be the binary operator $<$ defined over the event's attribute $Timestamp$, hence $\forall e_i, e_j \in E, e_i < e_j$ iff $Timestamp_{e_i} < Timestamp_{e_j}$. A simple consistency property could be *"A mobile phone connected to cell A, cannot connect to a different cell unless it disconnects from A"*. Finally, the finite horizon can be set to the maximum dataset cardinality, namely $T = \max_{S_i \subseteq S} |S_i|$.

We can model this consistency property using an FSS. The consistency model is graphically represented in Fig. 1(a). The system state is composed of two variables, namely (1) the variable *cell*, which describes to which cell the mobile phone is connected and (2) the variable $state \in \{con, dis\}$, whereas *con* denotes a phone connected to a cell, *dis* otherwise.

The data source S can be a database instance (e.g., an actual FSEDB) to be verified against the consistency model. Note that for each different S_i (i.e., for each different FSED) the model checker generates a different FSS modelling the S_i consistency. In other words, each different FSS is instantiated according to the actual data, therefore each FSS will have its own state space.

To clarify this aspect, it is worth to describe a different domain where a similar approach is used.

[5] Generally, researchers often use the term *model* referring to the result of a modelling phase (see, e.g. Baier et al. (2008)), despite two distinct tasks are involved, namely to (1) represent the system evolution (i.e., the model) and (2) formalise the properties to be verified.

Let us consider a discrete-time hybrid system[6] modelling a continuous domain where a car needs to cover a specified distance in the least possible time by incrementing or decrementing its current acceleration.

Clearly, due to the nonlinearity of the dynamics, discretising the time either every 10 seconds or every 5 seconds may generate different values for distance, velocity and acceleration, then resulting in *different state spaces*. As a consequence, the model evaluation through a model checker will generate different Finite State Systems according the discretisation time step chosen, see the Discretise and Validate approach of (Fox et al., 2012). Similarly, in our model, data (i.e., a S_i) play the same role that the time has in the car model: the model checker will generate and verify different FSSs according to the actual data. As a drawback, this behaviour prevents the identification of general patterns of data inconsistency, which could be useful for the purpose of generalizability. The following example should help in clarifying this concept.

Working Example. Let us consider again the Mobile Phone Tracking example of Tab. 1 and let us focus on the inconsistent event sequences of two mobile phones: $Mob_1 = (cell - in, 03290), (cell - out, 03291)$ and $Mob_2 = (cell - in, 03120), (cell - out, 03288)$, whereas respectively the $EType_i$ and $Cell - ID_i$ attributes only are reported, and very short sequences are showed for the sake of simplicity. The inconsistencies found share a common characteristic: the *"cell-out"* has been made on a cell different from the one where the last *"cell-in"* took place. We introduce the *data abstraction* to better manage such general inconsistencies. We replace the actual *cell* domain data $D_{cell} = \{03120, 03288, 03290, 03291, \ldots\}$ (whose cardinality can be very high although finite) with an abstract domain composed by a set of symbols. We can make an abstraction of the domain D_{cell} by using only a reduced set of symbols, namely $D_{cell}^{abstract} = \{C_X, C_Y\}$ as described next.

The idea to produce an abstract model from a concrete one is not new (also known as *abstract interpretation*, see (Cousot & Cousot, 1977)) and it has been applied also in the verification of transition systems (see the work of Clarke et al. (1994) where abstraction has been formally and widely addressed). In our context, the key intuition is that a relation between actual and abstract data can be based on an equivalence relation, thus an abstract state will represent several actual ones. This approach, in turn, generates an *abstract state space* which can be explored by an *abstract* consistency model, useful to generalise data inconsistency.

We formalise this concept as follows.

Definition 4 (Data Abstraction)**.** *Let s be an FSS state and e be an event with respectively $s = x_1, \ldots, x_n$ state variables and $e = (r_1, \ldots, r_m)$ event attributes. Let D be a finite (although very large) attribute domain where $\{x_1, \ldots, x_{n'}\} \subseteq \{x_1, \ldots, x_n\}$ and $\{r_1, \ldots, r_{m'}\} \subseteq \{r_1, \ldots, r_m\}$ are instances of D, i.e., $\{x_1, \ldots, x_{n'}\} \in D$ and $\{r_1, \ldots, r_{m'}\} \in D$.*

An event e happening in the state s requires the evaluation of $x_1, \ldots, x_{n'}$ and $r_1, \ldots, r_{m'}$ values, namely a configuration of $n' + m'$ different values of D. Then, we define the Abstract Domain *of D as a set of* different *symbols $d_1, \ldots, d_{n'+m'}$, called* Abstract Data*, required to represent the values of D in the consistency model, i.e. $D^{abstract} = \{d_1, \ldots, d_{n'+m'}\}$.*

Some conditions should be met to apply such data abstraction:
(p1) no total order relation is defined in the actual domain (or the total order relation is not considered for the scope of the analysis);
(p2) No condition should compare a symbol to a non-abstract value (e.g., $C_X = 03120$ in our example).

5. The Multidimensional Robust Data Quality Analysis

In this section we first introduce the Robust Data Quality Analysis, then we describe its extension, namely the Multidimensional Robust Data Quality Analysis, which identifies, extracts, and classifies data inconsistencies.

[6] A Discrete Time Hybrid System is a formal model for mixed discrete-continuous systems which allows the presence of both continuous and discrete variables. As a characteristic, it operates on a discrete state and performs discontinuous state changes at discrete time points.

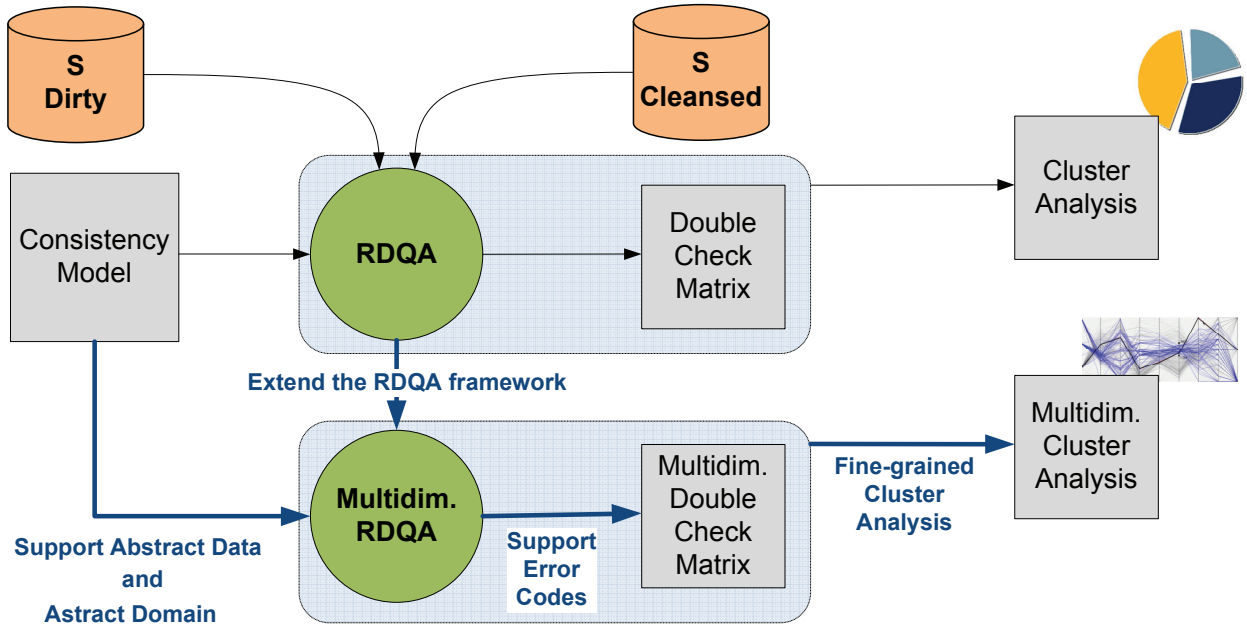


Figure 2: An overview of the RDQA and Multidimensional RDQA processes.

5.1. The RDQA

The Robust Data Quality Analysis (RDQA for short) is a model-based technique to verify the consistency of a longitudinal database before and after the cleansing intervention, as we discuss in the application domain of Sec. 7. Let us have and FSEDB S which can be decomposed in several FSED S_i as outlined in Def. 3. Let us introduce a function *clr* that can generate a cleansed (and consistent) version C_i of a source dataset S_i. In this respect, the *clr* cleansing function is considered regardless of its implementation and can be deemed as a black-box working as follows.

Function 1 (*clr*)**.** *Let S_i be an FSED, then clr : $FSED \rightarrow FSED$ is a function able to generate the cleansed instance C_i of a dataset S_i.*

Then, several questions arise about the *believability* of the cleansing process: what is the degree of consistency achieved through *clr*? Can we improve the consistency of the cleansed output? Are we sure that *clr* does not introduce any error in the cleansed dataset?

To answer these questions we need a function able to *check* the consistency of a dataset before and after the cleansing intervention, providing knowledge to evaluate the entire cleansing process.

Function 2 (*ccheck*)**.** *Let K_i be a FSED. Let $\sim$ be a* total order *relation such that $\epsilon \in K_i$ is an $\sim$-ordered event sequence, as defined in Def. 3.*

Then ccheck : $FSED \rightarrow \{0, 1\}$ is a function that returns 1 if ϵ is inconsistent, 0 otherwise.

Clearly, the *ccheck* can be realised by means of several programming paradigms. In this paper the *ccheck* has been implemented through UPMurphi, thus expressing the data consistency verification as a model checking problem, as described in Sec. 4.3.

Even though the *ccheck* is model-based, no guarantees are given about the correctness of *ccheck* (it is well-known that any model-based reasoning is only as good as the model is).

Given a FSED S_i and its cleansed instance C_i, since no details about the cleansing process are provided, we need to know the extent to which the *clr* has modified or not the output. We can guess whether the *clr* function has affected or not a dataset by comparing the source and cleansed instances (i.e, $S_i \stackrel{?}{=} C_i$). To this end, we introduce the following.

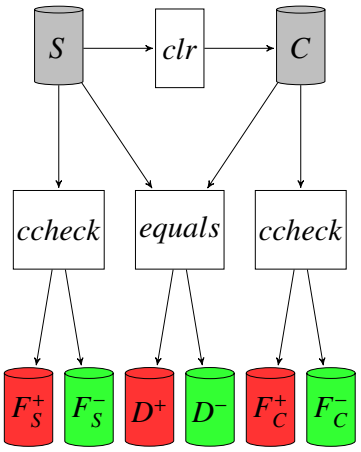


Figure 3: A schematic view of a single RDQA iteration

Function 3 (*equals*)**.** *Let S and C be FSEDs, we define equals : $FSED \times FSED \rightarrow \{0,1\}$ which returns 0 if no differences between S_i and C_i are found, 1 otherwise.*

The Fig. 2 describes the RDQA process which takes as input: a consistency model of the data, the source database S, and its cleansed instance C. The cleansed version C was generated executing the cleansing function *clr* on each $S_i \subseteq S$. A graphical representation of such approach is depicted in Fig. 1(b). Finally, for a given set S_i (C_i) we define a function returning the *representative element* of the set S_i (C_i).[7]

Function 4 (*rep*)**.** *Let K be an $FSEDB$ and let R be the set of all the representative elements of K. For all $K_i \subseteq K$ $rep : FSED \rightarrow R$ is a function which returns the representative element $r_i \in R$.*

The output of a RDQA iteration is the Double Check Matrix (DCM), as shown in Tab. 2(a) and 2(b), produced by collecting the results of functions *ccheck*, *equals*, and *clr* and computing statistics on the resulting S_i and C_i clusters. For the sake of clarity, we provide the pseudo-code of the RDQA in Procedures 1, 2, and 3. Furthermore, Fig. 3 provides a graphical overview of a single RDQA iteration.

To give a few example, the Row 1 (called also cluster 1) of Tab. 2(a) provides information about how many sequences have been considered consistent by both *clr* and *ccheck* functions and no differences between the original instance and the cleansed one has been found by *equals*. On the contrary, row 4 (called also cluster 4) represents the number of sequences for which no error was found by *ccheck* on the source, whereas the *equals* certifies that a cleansing intervention took place, and the sequence was considered inconsistent after cleansing. Finally, row 8 represents the number of sequences originally inconsistent that were modified during the cleansing with no success, since after the intervention they are still marked as inconsistent by the *ccheck*. The other cases will be extensively commented in Sec. 7 focusing on a specific application domain.

The DCM provides useful insights about the consistency of *clr* results and helps the identification of cleansing issues. The DCM results also contribute to the identification of errors in the formalisation of the consistency model, which in turn allows a better understanding of the domain rules. The RDQA procedure is applied iteratively by refining at each step the functions *clr* and *ccheck*. Clearly, this approach does not guarantee the correctness of the data cleansing process, nevertheless it helps making the process more robust with respect to data consistency.

5.2. The Multidimensional RDQA

Basically, the RDQA exploits the *ccheck* function to analyse the effectiveness of a cleansing routine *clr*. However, the *ccheck* function works according to an on/off approach: it can detect an inconsistency, but it does not provide any further information about the inconsistency characteristics.

[7] Intuitively, in a database record the representative element could be the primary key value or a hash value computed on the selected attributes.

Table 2: (a) The Double Check Matrix. (b) The explanation of the sets identified by the *ccheck* and the *equals* functions

(a)

Conditions			*Result*
$ccheck(S_i)$	$equals(S_i, C_i)$	$ccheck(C_i)$	Cardinality
0	0	0	$\lvert F_S^- \cap D^- \cap F_C^- \rvert$
0	0	1	$\lvert F_S^- \cap D^- \cap F_C^+ \rvert$
0	1	0	$\lvert F_S^- \cap D^+ \cap F_C^- \rvert$
0	1	1	$\lvert F_S^- \cap D^+ \cap F_C^+ \rvert$
1	0	0	$\lvert F_S^+ \cap D^- \cap F_C^- \rvert$
1	0	1	$\lvert F_S^+ \cap D^- \cap F_C^+ \rvert$
1	1	0	$\lvert F_S^+ \cap D^+ \cap F_C^- \rvert$
1	1	1	$\lvert F_S^+ \cap D^+ \cap F_C^+ \rvert$

(b)

$F_S^- = \bigcup(rep(S_i) \lvert ccheck(S_i) = 1)$
$F_S^+ = \bigcup(rep(S_i) \lvert ccheck(S_i) = 0)$
$F_C^+ = \bigcup(rep(C_i) \lvert ccheck(C_i) = 0)$
$F_C^- = \bigcup(rep(C_i) \lvert ccheck(C_i) = 1)$
$D^- = \bigcup(rep(S_i) \lvert equals(S_i, C_i) = 0)$
$D^+ = \bigcup(rep(S_i) \lvert equals(S_i, C_i) = 1)$

Procedure 1 RDQA

1. S =get_database_content();
2. $D^+ = \emptyset; D^- = \emptyset;$
3. $F_S^+ = \emptyset; F_S^- = \emptyset;$
4. $F_C^+ = \emptyset; F_C^- = \emptyset;$
5. **for all** $S_i \subseteq S$ **do**
6. $\quad C_i = clr(S_i);$
7. $\quad$ EQUALS_AUX(S_i, C_i);
8. $\quad$ CCHECK_AUX(S_i);
9. $\quad$ CCHECK_AUX(C_i);
10. **end for**
11. compute_DCM(); //Tab. 2(a)
12. display_DCM();

Procedure 2 EQUALS_AUX

Input: S_i, C_i

1. **if** $(equals(S_i, C_i) = 1)$ **then**
2. $\quad D^+ = D^+ \cup rep(S_i);$
3. **else**
4. $\quad D^- = D^- \cup rep(S_i);$
5. **end if**

Procedure 3 CCHECK_AUX

Input: X_i //It can be S_i or C_i

1. **if** $(ccheck(X_i) = 1)$ **then**
2. $\quad F_X^+ = F_X^+ \cup rep(X_i);$
3. **else**
4. $\quad F_X^- = F_X^- \cup rep(X_i);$
5. **end if**

To better clarify this concept, let us consider again the Mobile Phone Tracking example. In Tab. 3(b) we describe the system variables, the events, and the domain values used by a Finite State System modelling the expected system behaviour. The information can be used to identify in advance all the combination of <state, action> pairs leading to an inconsistent state. The set of <state, action> pairs can be reduced by using the abstract data set introduced in Sec. 4.3. According to the latter, the "cell" domain has been mapped on an abstract data set $D_{cell}^{abstract} = \{C_X, C_Y\}$. In this way a *Universal Checker* can be computed: an object which compactly represents all the possible inconsistencies which affect the data, as reported in Tab. 3(a)[8]. A taxonomy of errors can be obtained by assigning an error-code (i.e., a unique natural number) to each pair <state values; event values> leading to an inconsistency.

To this aim, in the following we introduce the concept of error-code on an event sequence and how to compute it on an FSS.

Definition 5 (Error-Code on FSED)**.** *Let S_i be an FSED and let $\epsilon = e_1, \ldots, e_n$ be a sequence of events according to Def. 3. Moreover, let i be the index of a minimal consistent subsequence, that is a sequence $\epsilon_i = e_1, \ldots, e_i$ such that ϵ_{i+1} is inconsistent while $\forall j : j \leq i \leq n-1$, ϵ_j is consistent.*

An error-code *err_ϵ for ϵ is a number $k \in \mathbb{N}^+$ such that $k = 0$ if ϵ_n is consistent, otherwise $k > 0$ uniquely identifies the inconsistency affecting the sequence ϵ_{i+1}.*

[8]Only a subset of the feasible pairs is reported, since the entries shown are reduced using some symmetry reduction techniques, e.g. $< state = con \wedge cell = C_X; cell - in, C_Y >$ and $< state = con \wedge cell = C_Y; cell - in, C_X >$ are symmetric then can be represented using only the first one according to the symmetry reduction technique (Norris Ip & Dill, 1996) present in UPMurphi.

Table 3: (a) Universal Checker for the Mobile Phone Tracking domain and (b) the values of its domain variables.

(a)

Err-Code	State	Incons. Event
1	$state$ = dis	$(cell-out, C_X)$
2	$state$ = dis	$(traffic, C_X)$
3	$state$ = con $\wedge$ cell = "C_X"	$(cell-out, C_Y)$
4	$state$ = con $\wedge$ cell = "C_X"	$(cell-in, C_X)$
5	$state$ = con $\wedge$ cell = "C_X"	$(cell-in, C_Y)$
6	$state$ = con $\wedge$ cell = "C_X"	$(traffic, C_Y)$

(b)

Variable Type	Variable	Domain Values
State Variables	state	con, dis
	cell	C_X, C_Y
Event data	cell	C_X, C_Y
	EType	cell-in, cell-out, traffic

Clearly, to compute error-codes on a FSED, we need to improve the *ccheck* function as follows.

Function 5 ($ccheck^{ec}$)**.** *Let K_i be a FSED. Let $\sim$ be a* total order *relation such that $\epsilon \in K_i$ is an $\sim$-ordered event sequence, as defined in Def. 3.*

Then $ccheck^{ec} : FSED \rightarrow \mathbb{N}$ is a function that returns an error-code err_ϵ, according to Def. 5, such that $err_\epsilon > 0$ if an inconsistency with error-code err_ϵ has been found on ϵ, 0 *otherwise.*

Also in this case, we used model-checking to implement the $ccheck^{ec}$ semantics. In particular, error-codes have been identified as follows.

Definition 6 (Error-Code on FSS)**.** *Let $\mathcal{S} = (S, I, A, F)$ be an FSS, let E be the set of errors states (i.e. inconsistent states) and T is the finite horizon. Moreover, let Π be the set of all the inconsistent trajectories $\pi \in \Pi$, i.e. $\pi = s_0 a_0 \ldots s_i a_i\ s_{i+1}$ with $s_0 \in I$, $|\pi| \leq T$ and $s_{i+1} \in E$. We introduce:*

- *the* error-code function $h : Reach(S) \times A \rightarrow \mathbb{N}^+$ *that, for a given pair (s_i, a_i) generates a unique natural number $h(s_i, a_i)$;* 0 *is used to denote a* no error *condition;*
- *a* Universal Checker $\mathcal{K}$ *as a map where each pair (s_i, a_i) is assigned to an error-code $h(s_i, a_i)$.*

Note that, the h function has been implemented by using the STL hash object class, since UPMurphi allows the use of external C/C++ libraries.

Roughly speaking, the Universal Checker represents a taxonomy of all the inconsistencies affecting the data. This, in turn, can be used to enhance a DCM cluster with error-code data as follows.

- Each DCM cluster is enriched with a square matrix having $n + 1$ rows and columns, where n is the number of distinct error-codes detected (i.e., $|\mathcal{K}|$) according to Def. 6, as shown in Eq. 1.

$$\forall l \in \{1, \ldots, 8\}\ M^l_{n+1,n+1} = \begin{pmatrix} err_{0,0} & err_{0,1} & \cdots & err_{0,n} \\ err_{1,0} & err_{1,1} & \cdots & err_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ err_{n,0} & err_{n,1} & \cdots & err_{n,n} \end{pmatrix} \tag{1}$$

- for a given cluster of the DCM, an element of the matrix $err_{i,j}$ is a positive number $k \in \mathbb{N}^+$ if k distinct event sequences have presented the error-code i in the original dataset and the error-code j in the cleansed one, otherwise k is zero. To give a few examples, $err_{0,0}$ is the number of sequences consistent before and after the cleansing intervention, while $err_{1,5}$ represents the number of sequences presenting the error-code 1 before the intervention and error-code 5 after the cleansing.

6. Experimental Phase: a Real-Life Application on the Labour Market Domain

In this section we introduce our application domain in the field of the Labour Market, describing how the data consistency problem has been described as a model checking one.

According to the Italian Labour Law, every time an employer hires or dismisses an employee, or an employment contract is modified (e.g. from part-time to full-time, or from fixed-term to unlimited-term) a communication (i.e., an event) is sent to a job registry. Those information are called *Mandatory Communications* (CO). From 1997, the Italian public administration has been using an Information System, called the *"CO System"* (The Italian Ministry of Labour and Welfare, 2012), where data concerning employment and active labour market policies are stored. Thus generating an administrative archive useful for studying the labour market dynamics. Indeed, extracting the longitudinal data by the CO archives allows one to observe the overall *evolution* of the labour market for a given observation period, obtaining insightful information about worker career paths, patterns and trends, facilitating the decision making processes of civil servants and policy makers, as studied by Cipollini et al. (2013); Lovaglio & Mezzanzanica (2013).

Unfortunately the CO archive data quality is very poor, and several studies have looked at the social and economic effects of inadequate data quality see, e.g. Haug et al. (2011); Redman (1998); Fisher & Kingma (2001); Wang & Strong (1996); Cesarini et al. (2007). The RDQA approach presented in the previous section has been used to assess and improve the data cleansing process performed on the mandatory communications of an Italian region inhabitants.

6.1. Domain Constraints

For each worker, a mandatory notification (an *event* in our context) is decomposed into the following attributes:

w_id: id identifying the person involved in the event;

e_id: id identifying the event;

e_date: event occurrence date;

e_type: type of events affecting the worker career. Events types are the *start* or the *cessation* of a working contract, the *extension* of a fixed-term contract, or the *conversion* from a contract type to a different one;

c_flag: specifies whether the event is related to a full-time or a part-time contract;

c_type: contract type under the Italian law (e.g. fixed-term or unlimited-term contract, etc.).

empr_id: employer involved in the event.

The development of a consistent career path over time is described by an *event sequence* ordered with respect to *e_date*. More precisely, in these settings an FSED is the (ordered) set of events for a given *w_id* (i.e. an FSED represents a single career in this context), and the FSEDs union composes the FSEDB.

Now we look more closely at the careers consistency, where the consistency semantic is derived from the Italian labour law, from the domain knowledge, and from the common practice. Some domain constraints are briefly reported:

c1: an employee can have no more than one full-time contract in force at any given time;

c2: an employee cannot have more than K part-time contracts (signed by different employers); in our context we assume $K = 2$ i.e., an employee cannot have more than two part time jobs active at the same time;

c3: an *unlimited term* contract cannot be extended;

c4: a contract extension cannot change the existing contract type (c_type) and the part-time/full-time status (c_flag) e.g., a part-time fixed-term contract cannot be turned into a full-time contract by an extension;

c5: a conversion requires either the c_type or the c_flag to be changed (or both).

For simplicity, we omit to describe some trivial constraints e.g., an employee cannot have a *cessation* event for a company for which she/he does not work, an event cannot be recorded twice, etc.

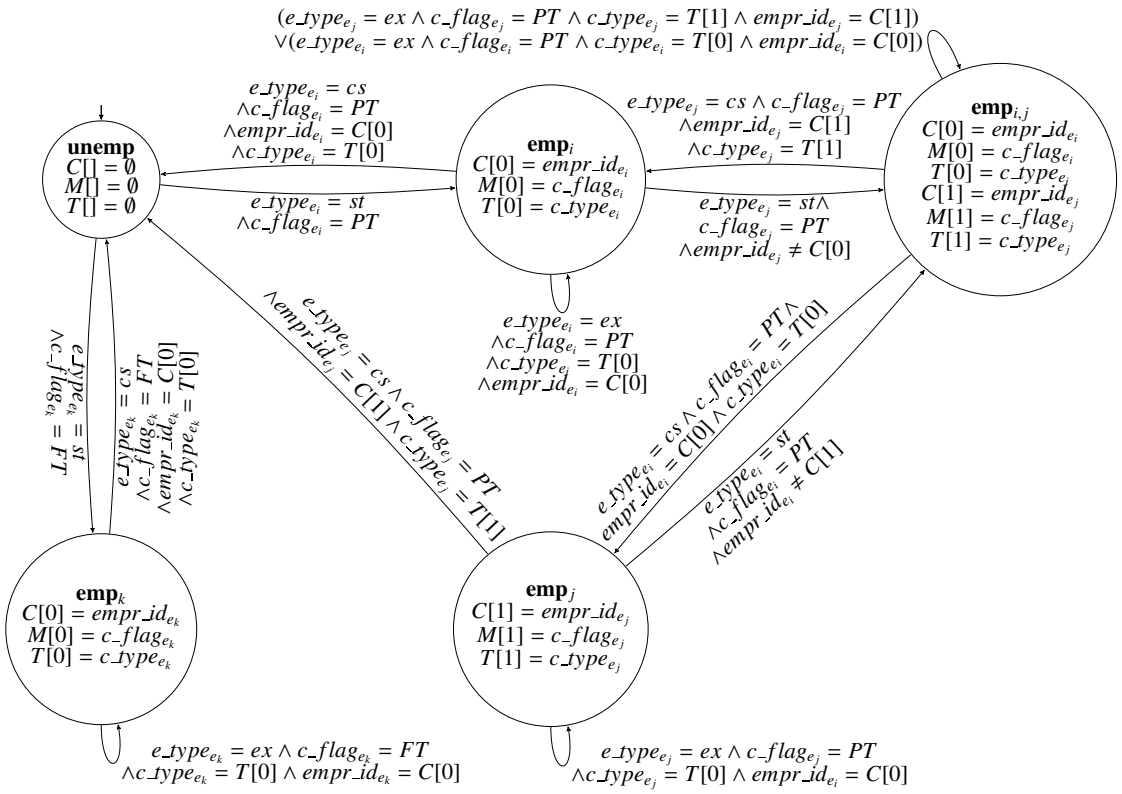


Figure 4: A graphical representation of a valid worker's career FSS where *st* = *start*, *cs* = *cessation*, *cn* = *conversion* and *ex* = *extension*.

6.2. Domain Modelling

We used UPMurphi to build the consistency model and to verify the data consistency. A worker's career at a given time point (i.e., the system state) is composed by three elements: the list of companies for which the worker has an active contract ($C[]$), the list of modalities (part-time, full-time) for each contract ($M[]$) and the list of contract types ($T[]$).

To give an example, $C[0] = 12,\ M[0] = PT,\ T[0] = unlimited$ models a worker having an active unlimited part-time contract with company 12.

A graphical representation of the domain model is showed in Figure 4 and it outlines a consistent career evolution. Note that, to improve the readability, we omitted to represent *conversion* events as well as inconsistent states/transitions (e.g., a worker activating two full-time contracts), which have been considered into the UPMurphi model. A valid career can evolve signing a part-time contract with company i, then activating a second part-time contract with company j, then closing the second part-time, and then reactivating the latter again (i.e., $unemp, emp_i, emp_{i,j}, emp_i, emp_{i,j}$).

For the sake of completeness, in Fig. A.7 and A.8 we provide the UPMurphi model for our application domain.

Finally, a mapping from actual to abstract data has been identified as described in Sec 4.3 taking into account both states and events of the automaton of Fig. 4. The *empr_id* attribute domain has been mapped on a set of 3 symbols $\{empr_x, empr_y, empr_z\}$ according to Def. 4. The attribute domains of *c_type*, *e_type*, and *c_flag* have not been replaced by abstract data since their domains are already bounded.

Indeed, we highlight that the model of Fig. 4 satisfies conditions p1 and p2, namely: (1) a total order relation for the *empr_id* domain is defined but it is not considered in the automaton, and (2) there are no conditions comparing an abstract value with a non abstract one.

7. Experimental Results

In this section we describe some experimental results obtained on the Labour Market Domain presented in the previous section. The experimental setup is detailed in Sec. 7.1. Then we present the output of the RDQA and MRDQA iterations in Sec. 7.2 and Sec. 7.3 respectively. Some comments about the effectiveness of the applied techniques are outlined in Sec. 7.4 while Sec. 7.5 reports details about the online dataset.

7.1. Experimental Settings

We performed an extensive experimental evaluation of our approach on an administrative archive of an Italian Region composed by $21,805,653$ mandatory communications. The source archive (S from now on) contains data on the careers of $2,498,615$ people observed starting from 1^{st} January 2004 to 31^{st} December 2011. For each career a subset S_i is identified (where $i \in [1 \dots 2,498,615]$). In these settings, S_i is a $FSED$ while S is the $FSEDB$.

The cleansed FSEDB C has been generated by using the function *clr*, implemented exploiting ETL business rules developed through the Talend tool [9] at CRISP Research Centre (CRISP Research Centre Web Page, 2013) whilst *ccheck* function has been realised using the UPMurphi tool. The computation of a MRDQA iteration was performed on a 32 bits 2.2Ghz CPU (connected to a MySQL server through ODBC driver) in about 2 hours. Finally, for the sake of completeness, in the appendix we report an extract of the UPMurphi model used.

We highlight that the aim of this experimental section is (1) to verify the effectiveness (if any) of the proposed approach on a real-world domain instance and (2) to provide a fine-grained analysis of both the source dataset and the cleansing procedures, allowing domain experts to derive a large number of analysis, statistics and action points to improve the overall data quality process.

7.2. Results: the RDQA

Table 4 shows the double check matrix (DCM) computed by the RDQA. Each DCM line shows the number of FSEDs (i.e. the number of careers) satisfying the properties of the corresponding cluster. Note that clusters labelled with ($*$) represent the job careers dropped by the *clr* (in spite of their consistency), since those careers refer to workers living in regions different from the reference one. Those events are therefore outside the scope of the analysis.

Table 4: The Double Check Matrix (with additional information on event data) computed on the careers data of an Italian Region. Remark: $ccheck(x) = 0$ means that x is consistent, inconsistent otherwise; $equals(S_i, C_i) = 0$ means S_i is equal to C_i, not equal otherwise.

DCM Clusters				*Careers Data*		*Events Data*		
Cluster	$ccheck(S_i)$	$equals(S_i, C_i)$	$ccheck(C_i)$	#Careers	%	# Events	$avg(\|S_i\|)$	%
1	0	0	0	833,136	33.34	1,783,682	2.14	8.18
2	0	0	1	0	0.00	0	0.00	0.00
3	0	1	0	125,779	5.03	827,505	6.58	3.79
4	0	1	1	100	0.00	706	7.06	0.00
*	0	1	null	4,001	0.16	10,729	2.68	0.05
5	1	0	0	0	0.00	0	0.00	0.00
6	1	0	1	13,737	0.55	57,681	4.20	0.26
7	1	1	0	1,395,682	55.86	16,832,031	12.06	77.19
8	1	1	1	122,000	4.88	2,260,593	18.53	10.37
*	1	1	null	4,180	0.17	32,726	7.83	0.15

Each DCM row shown in Tab. 4 can be shortly commented as follows, by focusing on the columns *DCM clusters* and *Careers Data*:

Cluster 1: represents careers *already clean* that have been left *untouched* by *clr*. It provides an estimation of the *consistent careers* in the source archive.

[9] `http://www.talend.com/`

Table 5: Composition of Case 3 of Table 4

Case 3	# Careers	% Careers
Total	124,779	5.03
Only Intervention 1	93,459	74.9
Only Intervention 2	31,094	24.92
Both Intervention 1 and 2	224	0.18
Other Interventions	450	0.36

Cluster 2: refers to careers considered consistent (by *ccheck*) before but not after the cleansing, although they have not been touched by *clr*. As expected this subset is empty.

Cluster 3: describes consistent careers that have been *unexpectedly changed* by the *clr*. Note that, although such kind of careers remained consistent after the intervention, the *clr* behaviour was investigated to understand and clarify the role of the cleansing procedure. By inspecting this cluster the domain experts discovered that the *clr* implementation improperly changed events and values for some kind of careers. Since up to 5% of the overall careers are affected, this issue cannot be neglected. Two main intervention types are performed by the *clr* on cluster 3 careers. The sizes of the affected sets are summarised in Tab. 5. The domain experts identified that both *intervention 1* and *2* are wrong with respect to the expected semantics (although both produce consistent results), therefore the *clr* needs to be fixed. The remaining 0.36% of *other* interventions is actually under investigation (in Tab. 5), and this bears a witness to the finely grained analysis obtained through the RDQA.

Cluster 4: represents careers originally consistent that *clr* made inconsistent. These careers strongly helped identifying and correcting bugs in the *clr* implementation.

Cluster 5: refers to careers considered inconsistent (by *ccheck*) before but consistent after cleansing, although they have not been touched by *clr*. This subset is empty, as expected.

Cluster 6: describes inconsistent careers, that *clr* was unable to detect (and thus to correct), therefore they were not touched.

Cluster 7: describes the number of (originally) inconsistent careers that *ccheck* recognises as *properly cleansed* by *clr* at the end.

Cluster 8: represents careers originally inconsistent that have been *not properly cleansed* since, despite an intervention of *clr*, the function *ccheck* still identifies them as inconsistent.

The *"Careers Data"* columns show statistics focusing on the career number dimension. One can observe that the number of (initially) consistent careers in the source dataset (Cluster 1) is 33.34%. The cleansing routines were not able to recognise the 0.55% of the careers as inconsistent (Cluster 6). On the contrary, it recognised but not fixed 4.88% of the overall careers set (Cluster 8). The number of consistent careers reached after the *clr* intervention is 89.2% (the sum of clusters 1 and 7).

The DCM provides relevant information to evaluate the cleansing activities, shedding light on different behaviour of the consistency semantics implemented in *ccheck* and *clr*. To give a few examples, cases 2, 4, and 5 can denote a potential issue that domain experts should address.

In the next section we show how the MRDQA, as introduced in Sec. 7.3, has been applied to obtain fine-grained information on how improve the cleansing function.

7.3. Results: The Multidimensional RDQA

The multidimensional RDQA described in Sec. 5.2 has been used to deeply analyze the cleansing activities previously outlined in Sec. 7.1 and Sec. 7.2. The consistency model described in Fig. 4 and the abstract data described in Sec. 6.2 have been used to generate a Universal Cleanser.

As a result, 231 error-codes have been identified. Fig. 5 shows the results of the DCM most significant clusters, namely the clusters 6, 7, 8, and the second (*). On the x-axis we report the labels of the inconsistency patterns (i.e.

Table 6: Description of the DCM more relevant error codes

Error Code	State	(Inconsistent) Event Received	Cardinality
215	@emp_k(M:FT,T:Unlimit,C:CompX)	(start,FT,Limit,CompX)	357,188
214	@emp_k(M:FT,T:Unlimit,C:CompX)	(start,FT,Limit,CompY)	61,139
213	@emp_k(M:FT,T:Unlimit,C:CompX)	(start,FT,Unlimit,CompX)	102,431
212	@emp_k(M:FT,T:Unlimit,C:CompX)	(start,FT,Unlimit,CompY)	10,520
193	@emp_k(M:FT,T:Unlimit,C:CompX)	(cessation,FT,Limit,CompX)	10,644

the error codes). For each error code, the number of affected careers is reported (the triangle which refers to the value of the right y-axis). The histogram shows the distribution of the errors among the clusters: the left y-axis reports the distribution of careers as percentage among the DCM clusters.

The Multidimensional DCM represents a fine-grained analysis of both the source dataset and the cleansing procedures, which allows domain experts to perform several analysis and statistics to understand and improve the overall data quality process.

In this regard, we can identify several issues worth of intervention (**action points**). Roughly speaking, action points can be seen as a checklist that should be beneficial for domain expert and cleansing architects to improve the cleansing routines. We have identified, among the others, the following action points.

- Let us consider the error codes from 212 to 215, as shown in Tab. 6 (note that the closer the error codes, the similar the error patterns). All these inconsistencies emerge when a worker receives a *start* event while having an on-going full-time contract referring to the same employer. The several error codes differs for some event attributes, e.g. the limited and unlimited term contract flag. This information proves, as expected, that a huge number of closing contract communications are missing from the archive. Due to the high error occurrence (i.e., about 17% of total careers), the domain experts and the cleansing designers concluded that it is worth to pay attention to all the cleansing routines dealing with such cases.
- The Cluster 6 discovery careers where the *clr* failed the cleansing intervention. The error code 193 is the most frequent (i.e., 77% of careers in the cluster are affected by this error) and it refers to workers having an unlimited-term contract and receiving a limited-term cessation event. Fixing the error code 193 can dramatically reduce the cardinality of this group.
- Focusing only on clusters where the *clr always* failed the cleansing process, the error codes 93, 115, and 184 can be neglected as they represents only 4 careers. Differently, domain experts and cleansing designers have focused on the most frequent error-codes affecting Cluster 8 to identify bugs on cleansing routines.
- Finally, the inconsistencies are mainly generated when a worker is in a full-time or unemployed status (consider that 78% of the source dataset is composed by full-time events). Thus, all the cleansing routines related to full-time events should be widely analysed.

7.4. Result Comments

Thanks to the DCM and the further analysis on the error set generation we can summarise our results by computing the following indicators:

The consistency degree (Cluster 1+Cluster 4) of the source dataset before and after the cleansing intervention. Note that 33.34% of the careers are consistent. This result is enough to stress the significance of a cleansing process before using the data for decision making purposes.

The room for improvement (Cluster 3 + Cluster 4 + Cluster 6 + Cluster 8) represents cases where the cleansing intervention was not successful. These clusters account for 10.47% of the careers, and this value is a quantitative estimation about how the *clr* process could be improved and refined.

The quality improvement (Cluster 7-Cluster 4) achieved by the cleansing intervention accounts for 55.86% of the careers. Note that the use of a model-based approach to evaluate the cleansing process makes more reliable this value. In other words, the results obtained can be considered a measure of the *clr* effectiveness.

Figure 5: The error codes and the DCM clusters for Tab. 4. The x-axis report the labels of the inconsistency patterns (i.e. the error codes), the triangle (referring to the right y-axis) reports the number of careers belonging to the DCM clusters 6,7,8 and the second (*), whilst the left y-axis reports the distribution of careers as percentage among the DCM clusters.

The Action Points. The capability to identify error patterns affecting the data, their distribution and characteristics is a useful swiss army knife during the development of cleansing routines. Indeed, one can discover a set of issues and their relevance by analysing the DCM and the error codes distribution, and subsequently refining the cleansing routines, comparing previous and further versions, making more robust the data quality process.

7.5. The Online Dataset

A source archive containing $1,248,814$ mandatory communications describing $214,429$ careers extracted from the dataset presented in Sec. 7 has been made publicly available for download[10]. On such dataset we performed the consistency verification as detailed in the paper. The dataset is composed by the following tables:

The Worker Careers. It is a table composed by 7 columns, whose semantics has been detailed in Sec. 6.1.

The Consistency Verification Results. It is a table composed by three columns, namely the *worker id*, the *error code* and the *error index* of the event after the shortest consistent subsequence: Considering a career composed by n events, an error index i with $0 \leq i < n$ means that $i - 1$ events make the career consistent whilst the i-th event makes it inconsistent.

7.5.1. Experimental Results on Online Dataset

Table 7: The Double Check Matrix computed on the careers data of an Italian Region.

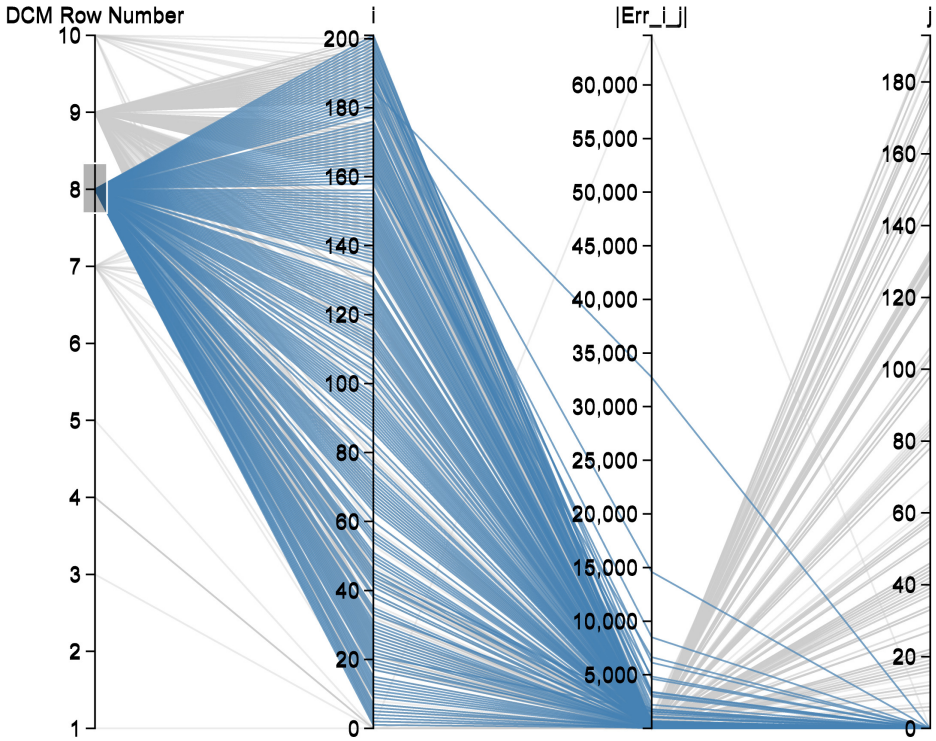
Row Number	*Cluster*	*DCM Cluster*			*Careers Data*	
		$ccheck(S_i)$	$equals(S_i, C_i)$	$ccheck(C_i)$	#Careers	%
R1	1	0	0	0	64,625	32.3
R2	2	0	0	1	0	0.00
R3	3	0	1	0	3,190	1.6
R4	4	0	1	1	184	0.09
R5	*	0	1	null	315	0.15
R6	5	1	0	0	0	0.00
R7	6	1	0	1	1,054	0.52
R8	7	1	1	0	116,216	58.1
R9	8	1	1	1	14,059	7.02
R10	*	1	1	null	357	0.17

The results of a single iteration of the RDQA are shown in Tab. 7. Finally, we also provide the result of the Multidimensional RDQA, by exploiting a well-known multidimensional visualisation technique, namely the *parallel-coordinates* (abbrv: ||-coord see Inselberg (1985)).

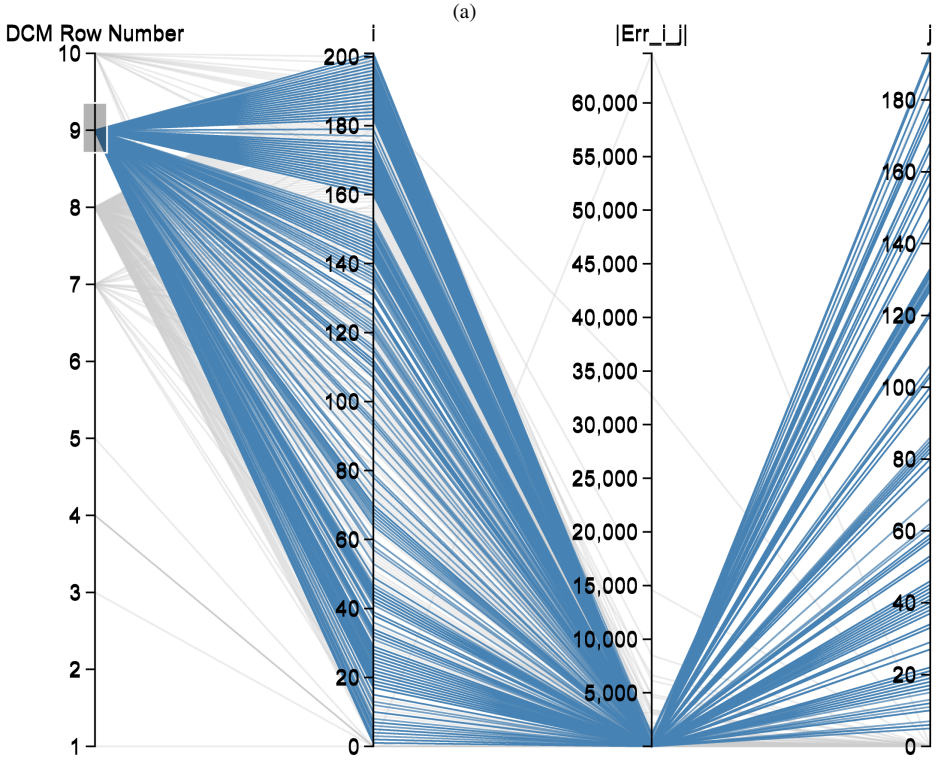
Informally speaking, ||-coord allow one to represent an n-dimensional datum $(x_1, \ldots, x_n)$ as a polyline, by connecting each x_i point in n parallel y-axes. We used the ||-coord to plot the DCM and the error-code data by using four dimensions, namely $(l, i, err_{i,j}, j)$ which respectively represent the DCM row number, the error-code before the cleansing, the number or careers (i.e. the $err_{i,j}$ value in Eq. 1), and the error-code after the cleansing. Generally, ||-coord tools show their powerfulness when used interactively (i.e., by selecting ranges from the y-axes, by emphasising the lines traversing through specific ranges, etc). For these reasons, the plot file has been made publicly available for downloading.

For the sake of completeness, we report two ||-coord graphs shown in Fig. 6. The Figure 6(a) shows the Cluster 7 of the DCM (i.e., the originally inconsistent careers correctly cleansed by the *clr* function). The figure explains how the error-codes are distributed on the original inconsistent data and their related frequencies. Differently, Fig. 6(b) highlights the Cluster 8 results (i.e. the careers improperly cleansed by the *clr*), which help in discovering how an error-code in the source dataset evolves due to a wrong cleansing intervention. Note that, in our experience, such

[10] Publicly available at `http://goo.gl/sS3rvv`



(a)



(b)

Figure 6: Parallel Coordinates for (a) Row Number 8 and (2) Row Number 9 of the DCM in Tab. 7. An online-demo is available at `http://goo.gl/sS3rvv`

information has played a key role in discovering the cleansing issues (e.g., it was easy to identify the most relevant and more numerous cases and to prioritise their fixing).

Finally, the Multidimensional RDQA outcomes and the ||-coord sheets have been made publicly available for download and demonstration, so that the results we present can be assessed, shared, and used by the community.

8. Conclusions and Expected Achievements

In this paper we have shown how the data quality tasks of a KDD process can be expressed as a model checking problem. Then, we presented the Multidimensional Robust Data Quality Analysis, an iterative and domain-independent technique aimed to (1) analyse the consistency of a dataset before and after the cleansing intervention and (2) iteratively improve the cleansing procedures by identifying the issues to be addressed. We implemented the consistency check through the UPMurphi tool (whose model is provided in the Appendix A) and we applied the MRDQA to a real-world governmental application in the field of the Italian Labour Market domain, dealing with the weakly structured data of million of citizens living in an Italian Region. Moreover, an anonymized instance of the dataset and the results have been made public available, in order to provide a dataset for KDD tasks to the community.

The traditional development of cleansing routines is a resource consuming and error prone activity as the huge set of data to be cleansed, the complexity of the domain, and the continuous business rules evolution make the cleansing process a challenging task. In such a scenario, the main benefits provided by our model-based approach for weakly-structured data are in (i) modelling data behaviour and quality constraints over time as a pathway on a graph, that allows expressing quality requirements that FDs and their variants do not handle, and (ii) the automatic evaluation of existing cleansing routines, that enables domain experts and decision makers to obtain insights about the dataset to be cleansed, reducing the human effort required for evaluating the realised cleansing intervention and improving the believability of the overall cleansing process as well.

More specifically, in our experience at the CRISP Research Centre[11], the results have pointed out the usefulness of our approach in supporting data cleansing tasks *before* using mining algorithms. To give a few examples, the MRDQA allowed domain experts and the cleansing architects to identify the initial database consistency degree (i.e., about 33%), the overall quality improvement obtained thanks to the use of their cleansing function (i.e., 55.86%) as well as the room for improvement of the cleansing function used (i.e., about 10%). Furthermore, the MRDQA has identified the distribution of the inconsistency patterns which, in turn supported the identification of several "action points", useful for investigating and improving the cleansing functions realised by means of an ETL tool.

Generally, speaking, the approach we presented enables domain experts to concentrate on *what* quality constraints need to be modelled rather than *how* to verify them, thus providing a (more) reliable cleansed data on which run mining techniques. Moreover, it simplifies the management of quality constraints by focusing on the model design, rather than working on the *engine* which catches them. Finally, it allows domain experts to discover quality patterns on data, helping them to achieve a better comprehension of data characteristics and dynamics.

Actually, we are working to expand our approach to perform cleansing by synthesising a *universal cleanser* (Mezzanzanica et al., 2013), i.e. an object which summarises the set of all cleansing actions for each feasible data inconsistency, according to a given consistency model by exploiting AI Planning languages and tool (Boselli et al., 2014b,c). As a further step, we have been working for evaluating the effectiveness of our approach on biomedical domain (Boselli et al., 2014a).

Acknowledgement

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions for improving this work.

[11] `www.crisp-org.it/en`

Appendix A. The UPMurphi Model

For the sake of completeness, in Fig. A.7 and A.8 we provide an extraction of the UPMurphi code for our application domain. The former contains the static part of the model, i.e., the system states as defined in Fig. 4 as well as the system procedure; the latter describes the system evolution (i.e., transitions) and invariant conditions, according to the automaton of Fig. 4. Note that the C/C++ functions handling the real-time connection with the MySQL Server have been wrapped by the function `nextEvent()`.

```
type
 companyT: Enum {emprX, emprY, emprZ};;
 dataT: 0..1000000;
 eventT: Enum {st,cs,cn,ex};
 jobT: Enum {PT,FT};
 contractT: Enum {Limited,Unlimited};
 stateT: Enum {null,unemp,emp_i,emp_j,emp_i_j,emp_k};
 activeJobT: 0..1;
 careerLengthT: -1..10000;
 error_state_code_T: stateT;
 event: Record -- it models an event
  e_type: eventT;
  empr_id: companyT;
  c_flag: jobT;
  c_type: contractT;
 end; -- Record.

var -- it describes the system state
 state: stateT;
 EOC:boolean; -- End Of Career
 e_iterator: careerLengthT;
 C: Array [activeJobT] of companyT;
 M: Array [activeJobT] of jobT;
 T: Array [activeJobT] of contractT;
 error_state: error_state_code_T;
 e: event;

-- nextEvent() retrieves data from the database
externfun nextEvent(it:careerLengthT):boolean "C_functions.h";

function in_error():boolean;
begin
 return error_state != null;
end;

procedure get_next();
begin
 if (!in_error()) then
  e_iterator := e_iterator + 1;
  EOC := nextEvent(e_iterator);
 end;
end;

procedure init_variables();
begin
 for j: activeJobT do
  C[j] := Undefined;
  M[j] := Undefined;
  T[j] := Undefined;
 end;
end;
```

```
/* Verify if a contract with company e.empr_id is still active.
 When "check_mod" is true it verifies also whether the
 contract is active with right Modality(M) and Type (T). */
function isActive(t:jobT; check_mod: boolean):boolean;
var found:boolean;
begin
 found := false;
 for j: activeJobT do
  if (C[j] = e.empr_id) then
   if (!check_mod & (t = M[j] | T[j] = e.c_type)) then
    found := true;
   elsif (check_mod & (t = M[j] & T[j] = e.c_type)) then
    found := true;
   endif;
  endif;
 end;
 return found;
end;

-- true if the contract conversion is allowed
function check_cn(index:activeJobT):boolean;
begin
 return isActive(M[index],false) & !(e.c_type = T[index] & M[
  index] = e.c_flag);
end;

-- true if the contract extension is allowed
function check_ex(index:activeJobT):boolean;
begin
 if (T[index] = Unlimited | e.c_type = Unlimited) then
  return false;
 endif;
 return isActive(M[index],false) & e.c_type = T[index] & M[
  index] = e.c_flag;
end;

startstate "generate the unemp state"
var temp:boolean;
begin
 state := unemp;
 e_iterator := 0;
 temp := nextEvent(0);
 init_variables();
 error_state := null;
 EOC := false;
end;
```

Figure A.7: the UPMurphi abstract model 1/2

```
-- modelling state unemp
rule "unemp"
state = unemp & !EOC & !in_error() ==>
begin
 if (e.e_type = st & e.c_flag = PT & !isActive(PT,false) & !
  isActive(FT,false)) then
  state := emp_i;
  C[0] := e.empr_id;
  M[0] := e.c_flag;
  T[0] := e.c_type;
 elsif (e.e_type = st & e.c_flag = FT & !isActive(PT,false) & !
  isActive(FT,false)) then
  state := emp_k;
  C[0] := e.empr_id;
  M[0] := e.c_flag;
  T[0] := e.c_type;
 else
  error_state := unemp;
 endif;
 get_next();
end;
-- modelling state emp_k
rule "emp_k"
state = emp_k & !EOC & !in_error() ==>
begin
 if (e.e_type = cs & e.c_flag = FT & isActive(FT,true)) then
  state := unemp;
  C[0] := Undefined;
  M[0] := Undefined;
  T[0] := Undefined;
 elsif (e.e_type = cn &check_cn(0)) then
  T[0] := e.c_type;
  M[0] := PT;
  state := emp_i;
 elsif (e.e_type = ex &check_ex(0)) then
  state := emp_k;
 else
  error_state := emp_k;
 endif;
 get_next();
end;
-- modelling state emp_i
rule "emp_i"
state = emp_i & !EOC & !in_error() ==>
begin
 if (e.e_type = st & e.c_flag = PT & !isActive(PT,false) & !
  isActive(FT,false)) then
  state := emp_i_j;
  C[1] := e.empr_id;
  M[1] := e.c_flag;
  T[1] := e.c_type;
 elsif (e.e_type = cs & e.c_flag = PT & isActive(PT,true)) then

  state := unemp;
  C[0] := Undefined;
  M[0] := Undefined;
  T[0] := Undefined;
 elsif (e.e_type = cn &check_cn(0)) then
  T[0] := e.c_type;
  M[0] := e.c_flag;
  if (M[0] = FT) then
   state := emp_k;
  end;
 elsif (e.e_type = ex &check_ex(0)) then
  state := emp_i;
 else
  error_state := emp_i;
 endif;
 get_next();
end;
```

```
-- modelling state emp_j
rule "emp_j"
state = emp_j & !EOC & !in_error() ==>
begin
 if (e.e_type = st & e.c_flag = PT & !isActive(PT,false) & !
  isActive(FT,false)) then
  state := emp_i_j;
  C[0] := e.empr_id;
  M[0] := e.c_flag;
  T[0] := e.c_type;
 elsif (e.e_type = cs & e.c_flag = PT & isActive(PT,true)) then

  state := unemp;
  C[1] := Undefined;
  M[1] := Undefined;
  T[1] := Undefined;
 elsif (e.e_type = cn &check_cn(1)) then
  if (e.c_flag = FT) then
   T[0] := e.c_type;
   M[0] := e.c_flag;
   C[0] := C[1];
   T[1] := Undefined;
   M[1] := Undefined;
   C[1] := Undefined;
   state := emp_k;
  else
   T[1] := e.c_type;
   M[1] := e.c_flag;
  end;
 elsif (e.e_type = ex &check_ex(1)) then
  state := emp_j;
 else
  error_state := emp_j;
 endif;
 get_next();
end;
-- modelling state emp_i_j
rule "emp_i_j"
state = emp_i_j & !EOC & !in_error() ==>
begin
 if (e.e_type = cs & e.c_flag = PT & isActive(PT,true)) then
  if (e.empr_id = C[0]) then
   state := emp_j;
   C[0] := Undefined;
   M[0] := Undefined;
   T[0] := Undefined;
  elsif (e.empr_id = C[1]) then
   state := emp_i;
   C[1] := Undefined;
   M[1] := Undefined;
   T[1] := Undefined;
  endif;
 elsif (e.e_type = ex & (check_ex(1) |check_ex(0))) then
  state := emp_i_j;
 elsif (e.e_type = cn &check_cn(0) & e.c_flag != FT ) then
  state := emp_i_j;
  T[0] := e.c_type;
 elsif (e.e_type = cn &check_cn(1) & e.c_flag != FT ) then
  state := emp_i_j;
  T[1] := e.c_type;
 else
  error_state := emp_i_j;
 endif;
 get_next();
end;

goal "verify consistency"
in_error();
```

Figure A.8: The UPMurphi abstract model 2/2

References

Abello, J., Pardalos, P. M., & Resende, M. G. (2002). *Handbook of massive data sets* volume 4. Springer.

Afanasiev, L., Franceschet, M., Marx, M., & de Rijke, M. (2004). CTL model checking for processing simple xpath queries. In *Proceedings of the 11th International Symposium on Temporal Representation and Reasoning (TIME)*. (pp. 117–124).

Arenas, M., Bertossi, L. E., & Chomicki, J. (1999). Consistent query answers in inconsistent databases. In *ACM Symp. on Principles of Database Systems* (pp. 68–79). ACM Press.

Atlee, J. M., & Gannon, J. (1993). State-based model checking of event-driven system requirements. *Software Engineering, IEEE Transactions on*, *19*, 24–40.

Baier, C., Katoen, J.-P. et al. (2008). *Principles of model checking* volume 26202649. MIT press Cambridge.

Ballou, D. P., & Tayi, G. K. (1999). Enhancing data quality in data warehouse environments. *Communications of the ACM*, *42*, 73–78.

Barateiro, J., & Galhardas, H. (2005). A Survey of Data Quality Tools. *Datenbank-Spektrum*, *14*, 15–21.

Bartolucci, F., Farcomeni, A., & Pennoni, F. (2012). *Latent Markov models for longitudinal data*. Boca Raton, FL: Chapman & Hall/CRC Press.

Batini, C., Cappiello, C., Francalanci, C., & Maurino, A. (2009). Methodologies for Data Quality Assessment and Improvement. *ACM Comput. Surv.*, *41*, 16:1–16:52.

Batini, C., & Scannapieco, M. (2006). *Data Quality: Concepts, Methodologies and Techniques*. Data-Centric Systems and Applications. Springer.

Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., & Schnoebelen, P. (2010). *Systems and software verification: model-checking techniques and tools*. Springer Publishing Company, Incorporated.

Bertossi, L. (2006). Consistent query answering in databases. *ACM Sigmod Record*, *35*, 68–76.

Biere, A., Cimatti, A., Clarke, E. M., Strichman, O., & Zhu, Y. (2003). Bounded model checking. *Advances in computers*, *58*, 117–148.

Boselli, R., Cesarini, M., Mercorio, F., & Mezzanzanica, M. (2013). Inconsistency knowledge discovery for longitudinal data management: A model-based approach. In *SouthCHI13 special session on Human-Computer Interaction & Knowledge Discovery, LNCS, vol. 7947*. Springer.

Boselli, R., Cesarini, M., Mercorio, F., & Mezzanzanica, M. (2014a). A policy-based cleansing and integration framework for labour and healthcare data. In *SOTA Survey. HCI-KDD Methods: Dealing with complex data* LNCS vol. 8401. Springer.

Boselli, R., Cesarini, M., Mercorio, F., & Mezzanzanica, M. (2014b). Towards data cleansing via planning. *Intelligenza Artificiale*, *8*.

Boselli, R., Mezzanzanica, M., Cesarini, M., & Mercorio, F. (2014c). Planning meets data cleansing. In *The 24th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI.

Bravo, L., Fan, W., Geerts, F., & Ma, S. (2008). Increasing the expressivity of conditional functional dependencies without extra complexity. In *Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE)* (pp. 516–525). IEEE.

Cesarini, M., Mezzanzanica, M., & Fugini, M. (2007). Analysis-sensitive conversion of administrative data into statistical information systems. *Journal of Cases on Information Technology*, *9*, 57–81.

Choi, E.-H., Tsuchiya, T., & Kikuno, T. (2006). Model checking active database rules under various rule processing strategies. *IPSJ Digital Courier*, *2*, 826–839.

Chomicki, J. (1995). Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Transactions on Database Systems (TODS)*, *20*, 149–186.

Chomicki, J., & Marcinkowski, J. (2005a). Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, *197*, 90–121.

Chomicki, J., & Marcinkowski, J. (2005b). On the computational complexity of minimal-change integrity maintenance in relational databases. In *Inconsistency Tolerance* (pp. 119–150). Springer volume 3300 of *Lecture Notes in Computer Science*.

Cimatti, A., Roveri, M., & Traverso, P. (1998). Automatic obdd-based generation of universal plans in non-deterministic domains. In J. Mostow, & C. Rich (Eds.), *AAAI/IAAI* (pp. 875–881). AAAI Press / The MIT Press.

Cipollini, F., Ferretti, C., Ganugi, P., & Mezzanzanica, M. (2013). A continuous time mover-stayer model for labor market in a northern italian area. In *Classification and Data Mining* (pp. 181–188). Springer.

Clarke, E. M., Grumberg, O., & Long, D. E. (1994). Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *16*, 1512–1542.

Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model Checking*. The MIT Press.

Clemente, P., Kaba, B., Rouzaud-Cornabas, J., Alexandre, M., & Aujay, G. (2012). Sptrack: Visual analysis of information flows within selinux policies and attack logs. In *AMT Special Session on Human-Computer Interaction and Knowledge Discovery* (pp. 596–605). Springer volume 7669 of *LNCS*.

Cong, G., Fan, W., Geerts, F., Jia, X., & Ma, S. (2007). Improving data quality: Consistency and accuracy. In *Proceedings of the 33rd international conference on Very large data bases* (pp. 315–326). VLDB Endowment.

Cousot, P., & Cousot, R. (1977). Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages* (pp. 238–252). ACM.

CRISP Research Centre Web Page (2013). http://www.crisp-org.it.

Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I. F., Ouzzani, M., & Tang, N. (2013). Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 international conference on Management of data* (pp. 541–552). ACM.

Dasu, T. (2013). Data glitches: Monsters in your data. In *Handbook of Data Quality* (pp. 163–178). Springer.

Della Penna, G., Intrigila, B., Magazzeni, D., & Mercorio, F. (2009). UPMurphi: a tool for universal planning on PDDL+ problems. In *Proceeding of the 19th International Conference on Automated Planning and Scheduling (ICAPS)* (pp. 106–113). AAAI Press.

Della Penna, G., Intrigila, B., Magazzeni, D., & Mercorio, F. (2010a). A PDDL+ benchmark problem: The batch chemical plant. In *Proceedings of ICAPS 2010* (pp. 222–224). AAAI Press.

Della Penna, G., Intrigila, B., Magazzeni, D., & Mercorio, F. (2010b). Planning for autonomous planetary vehicles. In *Proceedings of the The Sixth International Conference on Autonomic and Autonomous Systems* (pp. 131–136). Cancun, Mexico: IEEE.

Della Penna, G., Intrigila, B., Magazzeni, D., Mercorio, F., & Tronci, E. (2011). Cost-optimal strong planning in non-deterministic domains. In *Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics (ICINCO)* (pp. 56–66). SciTePress.

Della Penna, G., Intrigila, B., Melatti, I., Tronci, E., & Venturini Zilli, M. (2004). Exploiting transition locality in automatic verification of finite state concurrent systems. *STTT*, *6*, 320–341.

Della Penna, G., Magazzeni, D., & Mercorio, F. (2012). A universal planning system for hybrid domains. *Applied Intelligence*, *36*, 932–959.

Devaraj, S., & Kohli, R. (2000). Information technology payoff in the health-care industry: a longitudinal study. *Journal of Management Information Systems*, *16*, 41–68.

Dill, D. L. (1996). The mur ϕ verification system. In *Computer Aided Verification* (pp. 390–393). Springer.

Dovier, A., & Quintarelli, E. (2002). Model-checking based data retrieval. In *Database Programming Languages* (pp. 62–77). Springer Berlin / Heidelberg volume 2397 of *LNCS*.

Dovier, A., & Quintarelli, E. (2009). Applying Model-checking to solve Queries on semistructured Data. *Computer Languages, Systems & Structures*, *35*, 143 – 172.

Edelkamp, S., & Jabbar, S. (2006). Large-scale directed model checking ltl. In *Model Checking Software* (pp. 1–18). Springer.

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

European Telecommunications Standards Institute ES 201 671 (2009). Handover interface for the lawful interception of telecommunications traffic.

Fan, W. (2008). Dependencies revisited for improving data quality. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (pp. 159–170). ACM.

Fan, W., Li, J., Ma, S., Tang, N., & Yu, W. (2010). Towards certain fixes with editing rules and master data. *Proceedings of the VLDB Endowment*, *3*, 173–184.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, *39*, 27–34.

Fellegi, I., & Sunter, A. (1969). A theory for record linkage. *Journal of the American Statistical Association*, *64*, 1183–1210.

Fellegi, I. P., & Holt, D. (1976). A systematic approach to automatic edit and imputation. *Journal of the American Statistical association*, *71*, 17–35.

Fisher, C., Lauría, E., Chengalur-Smith, S., & Wang, R. (2012). *Introduction to information quality*. AuthorHouse.

Fisher, C. W., & Kingma, B. R. (2001). Criticality of data quality as exemplified in two disasters. *Inf. Manage.*, *39*, 109–116.

Fox, C., Levitin, A., & Redman, T. (1994). The notion of data and its quality dimensions. *Information processing & management*, *30*, 9–19.

Fox, M., Long, D., & Magazzeni, D. (2011). Automatic construction of efficient multiple battery usage policies. In T. Walsh (Ed.), *IJCAI* (pp. 2620–2625). IJCAI/AAAI.

Fox, M., Long, D., & Magazzeni, D. (2012). Plan-based policies for efficient multiple battery load management. *J. Artif. Intell. Res. (JAIR)*, *44*, 335–382.

Giunchiglia, F., & Traverso, P. (2000). Planning as model checking. *Recent Advances in AI Planning*, (pp. 1–20).

Greco, G., Greco, S., & Zumpano, E. (2001). A logic programming approach to the integration, repairing and querying of inconsistent databases. In *Logic Programming* (pp. 348–364). Springer.

Hansen, P., & Järvelin, K. (2005). Collaborative information retrieval in an information-intensive domain. *Information Processing & Management*, *41*, 1101–1119.

Haug, A., Zachariassen, F., & Van Liempd, D. (2011). The costs of poor data quality. *Journal of Industrial Engineering and Management*, *4*, 168–193.

Herrera-Viedma, E., & Peis, E. (2003). Evaluating the informative quality of documents in sgml format from judgements by means of fuzzy linguistic techniques based on computing with words. *Information Processing & Management*, *39*, 233–249.

Hipp, J., Güntzer, U., & Grimmer, U. (2001). Data quality mining-making a virute of necessity. In *DMKD*.

Holzinger, A. (2011). Weakly structured data in health-informatics: the challenge for human-computer-interaction. In *Proceedings of INTERACT 2011 Workshop: Promoting and Supporting Healthy Living by Desing* (pp. 5–7). IFIP.

Holzinger, A. (2012). On knowledge discovery and interactive intelligent visualization of biomedical data - challenges in human-computer interaction & biomedical informatics. In M. Helfert, C. Francalanci, & J. Filipe (Eds.), *DATA*. SciTePress.

Holzinger, A., Bruschi, M., & Eder, W. (2013a). On interactive data visualization of physiological low-cost-sensor data with focus on mental stress. In A. Cuzzocrea, C. Kittl, D. E. Simos, E. Weippl, & L. Xu (Eds.), *CD-ARES* (pp. 469–480). Springer volume 8127 of *Lecture Notes in Computer Science*.

Holzinger, A., Yildirim, P., Geier, M., & Simonic, K.-M. (2013b). Quality-based knowledge discovery from medical text on the web. In Pasi et al. (2013b).

Holzinger, A., & Zupan, M. (2013). Knodwat: a scientific framework application for testing knowledge discovery methods for the biomedical domain. *BMC Bioinformatics*, *14*, 191.

Holzmann, G. J., & Smith, M. H. (1999). A practical method for verifying event-driven software. In *Proceedings of the 21st international conference on Software engineering* (pp. 597–607). ACM.

Inselberg, A. (1985). The plane with parallel coordinates. *The Visual Computer*, *1*, 69–91.

Kahn, B. K., Strong, D. M., & Wang, R. Y. (2002). Information quality benchmarks: product and service performance. *Communications of the ACM*, *45*, 184–192.

Kolahi, S., & Lakshmanan, L. V. (2009). On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory* (pp. 53–62). ACM.

Kroening, D., Clarke, E., & Yorav, K. (2003). Behavioral consistency of C and Verilog programs using bounded model checking. In *Proceedings of DAC 2003* (pp. 368–371). ACM Press.

Levitin, A., & Redman, T. (1995). Quality dimensions of a conceptual view. *Information Processing & Management*, *31*, 81–88.

Lovaglio, P. G., & Mezzanzanica, M. (2013). Classification of longitudinal career paths. *Quality & Quantity*, *47*, 989–1008.

Madnick, S. E., Wang, R. Y., Lee, Y. W., & Zhu, H. (2009). Overview and framework for data and information quality research. *J. Data and Information Quality*, *1*, 2:1–2:22.

Mayfield, C., Neville, J., & Prabhakar, S. (2010). Eracer: a database approach for statistical inference and data cleaning. In *Proceedings of the*

2010 ACM SIGMOD International Conference on Management of data (pp. 75–86). ACM.

Mercorio, F. (2013). Model checking for universal planning in deterministic and non-deterministic domains. *AI Commun.*, *26*, 257–259.

Mezzanzanica, M., Boselli, R., Cesarini, M., & Mercorio, F. (2011). Data quality through model checking techniques. In *IDA* (pp. 270–281). Springer volume 7014 of *Lecture Notes in Computer Science*.

Mezzanzanica, M., Boselli, R., Cesarini, M., & Mercorio, F. (2013). Automatic synthesis of data cleansing activities. In M. Helfert, & C. Francalanci (Eds.), *The 2nd International Conference on Data Management Technologies and Applications (DATA)* (pp. 138 – 149). Scitepress.

Neven, F. (2002). Automata theory for xml researchers. *SIGMOD Rec.*, *31*, 39–46.

Newcombe, H. B., & Kennedy, J. M. (1962). Record linkage: making maximum use of the discriminating power of identifying information. *Communications of the ACM*, *5*, 563–566.

Norris Ip, C., & Dill, D. (1996). Better verification through symmetry. *Formal methods in system design*, *9*, 41–75.

Ferreira de Oliveira, M. C., & Levkowitz, H. (2003). From visual data exploration to visual data mining: A survey. *IEEE Trans. Vis. Comput. Graph.*, *9*, 378–394.

Pasi, G., Bordogna, G., & Jain, L. C. (2013a). An introduction to quality issues in the management of web information. In Pasi et al. (2013b).

Pasi, G., Bordogna, G., & Jain, L. C. (Eds.) (2013b). *Quality Issues in the Management of Web Information* volume 50 of *Intelligent Systems Reference Library*. Springer.

Pipino, L. L., Lee, Y. W., & Wang, R. Y. (2002). Data quality assessment. *Communications of the ACM*, *45*, 211–218.

Prinzie, A., & Van den Poel, D. (2011). Modeling complex longitudinal consumer behavior with dynamic bayesian networks: an acquisition pattern analysis application. *Journal of Intelligent Information Systems*, *36*, 283–304.

Rahm, E., & Do, H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, *23*, 3–13.

Ray, I., & Ray, I. (2001). Detecting termination of active database rules using symbolic model checking. In *Proceedings of the 5th East European Conference on Advances in Databases and Information Systems* ADBIS '01 (pp. 266–279). London, UK: Springer-Verlag.

Redman, T. C. (1998). The impact of poor data quality on the typical enterprise. *Commun. ACM*, *41*, 79–82.

Redman, T. C. (2001). *Data quality: the field guide*. Digital Pr.

Sadiq, S. (2013). *Handbook of Data Quality*. Springer.

Schoppers, M. (1987). Universal plans of reactive robots in unpredictable environments. In *Proc. IJCAI 1987.*.

The Italian Ministry of Labour and Welfare (2012). Annual report about the CO system, available at `http://www.cliclavoro.gov.it/Barometro-Del-Lavoro/Documents/Rapporto_CO/Executive_summary.pdf`.

Thomsen, C., & Pedersen, T. B. (2005). A survey of open source tools for business intelligence. In *Data Warehousing and Knowledge Discovery* (pp. 74–84). Springer.

Vardi, M. (1987). Fundamentals of dependency theory. *Trends in Theoretical Computer Science*, (pp. 171–224).

Wang, R. Y., & Strong, D. M. (1996). Beyond accuracy: What data quality means to data consumers. *J. of Management Information Systems*, *12*, 5–33.

Winkler, W. E. (1997). *Editing discrete data*. Bureau of the Census.

Winkler, W. E. (2000). Machine learning, information retrieval and record linkage. In *Proc Section on Survey Research Methods, American Statistical Association* (pp. 20–29).

Winkler, W. E. (2004). Methods for evaluating and creating data quality. *Information Systems*, *29*, 531–550.

Wong, B. W., Xu, K., & Holzinger, A. (2011). Interactive visualization for information analysis in medical diagnosis. In *Information Quality in e-Health* (pp. 109–120). Springer.

Yakout, M., Berti-Équille, L., & Elmagarmid, A. K. (2013). Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *Proceedings of the 2013 international conference on Management of data* (pp. 553–564). ACM.

Yu, L., Wang, S., & Lai, K. K. (2006). An integrated data preparation scheme for neural network data analysis. *Knowledge and Data Engineering, IEEE Transactions on*, *18*, 217–230.