Automatic Synthesis of Data Cleansing Activities

Mario Mezzanzanica, Roberto Boselli, Mirko Cesarini, and Fabio Mercorio
Department of Statistics and Quantitative Methods - C.R.I.S.P. Research Centre, University of Milan-Bicocca, Milan, Italy
{*firstname.lastname*}*@unimib.it*

Keywords: Data Quality, Data Management, Cleansing Algorithms, Model-based Reasoning

Abstract: Data cleansing is growing in importance among both public and private organisations, mainly due to the relevant amount of data exploited for supporting decision making processes. This paper is aimed to show how model-based verification algorithms (namely, model checking) can contribute in addressing data cleansing issues, furthermore a new benchmark problem focusing on the labour market dynamic is introduced. The consistent evolution of the data is checked using a model defined on the basis of domain knowledge. Then, we formally introduce the concept of *universal cleanser*, i.e. an object which summarises the set of all cleansing actions for each feasible data inconsistency (according to a given consistency model), then providing an algorithm which synthesises it. The universal cleanser can be seen as a repository of corrective interventions useful to develop cleansing routines. We applied our approach to a dataset derived from the Italian labour market data, making the whole dataset and outcomes publicly available to the community, so that the results we present can be shared and compared with other techniques.

1 INTRODUCTION

In the last two decades, the diffusion of Informative Systems has increased at an explosive rate, contributing to the definition and realisation of many IT services, also in the public sector. As a consequence, the amount of data that organisations are now handling is growing apace. Such data can contribute to analyse, observe and explain social, economic and business phenomena, as well as to assess decision making activities, e.g. the evaluation of active policies, resource allocation, service design and improvement. However, it is well known that the quality of the data is frequently very low (Fayyad et al., 2003) and, due to the "garbage in, garbage out" principle, dirty data strongly affect the information derived from them. Hence, data cleansing is a mandatory step before using data for decision making purposes.

Data quality and cleansing issues have been addressed in many fields of the literature, by dealing with several quality dimensions, see (Batini and Scannapieco, 2006). Here we focus on *consistency*, which takes into account the violation of semantic rules defined over a set of data items. This work concentrates on information about a given subject, object or phenomena, observed at multiple sampled time points: the result is a longitudinal dataset, also known as panel data, see (Singer and Willett, 2003; Bartolucci et al., 2012) for details, which allows one to study how data change along the time.

To this regard, let us consider the dataset showed in Tab. 1 modelling a cruise ship travel plan, as presented by (Mezzanzanica et al., 2012). The ship travels by sea and stops at the port of calls (intermediate destinations). The harbour regulations require a notification prior to entry into port. In this scenario, we suppose that a ship is required to perform a *checkin* notification when entering a harbour and a *checkout* when exiting. Looking at Tab. 1, one can note that the departure date from Lisbon is missing, since a checkout from Lisbon should be done prior to entering in Barcelona. In this sense, the events sequence modelling the travel of the ship S01 can be considered as inconsistent.

Table 1: Travel Plan of a Cruise Ship

ShipID	City	Date	Event Type
S01	Venice	12^{th} April 2011	checkin
S01	Venice	15^{th} April 2011	checkout
S01	Lisbon	30^{th} April 2011	checkin
S01	Barcelona	5^{th} May 2011	checkin
S01	Barcelona	8^{th} May 2011	checkout
...	...	...	...

One can argue that ships are usually moored in the harbour for 3 days, hence a cleansing activity could set the missing departure date from Lisbon on

the 3^{rd} May. Unfortunately, there is no certainty of having guessed the real value, and having a consistent dataset is required to obtain effective statistics (e.g., missing dates may have unpredictable effects when computing an indicator like *active travel days*/*overall cruise duration*).
The aims of this work are twofold: (1) we describe how a model-based reasoning (i.e., model checking) can be used to describe and verify the consistent evolution of the data along the time. Then, we show how such model can be exploited to synthesise a *Universal Cleanser*: an object summarising the set of *all* feasible cleansing actions for each feasible data inconsistency; (2) We present a real-world problem in the labour market context, providing both the source datasets and the results publicly available to the community, so that the data can be shared and compared with other studies.

2 RELATED WORK

Data quality and cleansing problems have widely been addressed, a lot of works cross the boundaries of different research fields, consequently it is not an easy task framing them into a holistic classification. Furthermore, there is no commonly agreed formal definition of data cleansing (Maletic and Marcus, 2010).

To the best of our knowledge not any other works deal with database consistency issues related to (arbitrarily long) sequences. Several existing works focus on constraint among attribute sets (single tuple scope), others concentrate on entity resolution problems that requires a pairwise comparison (two tuples scope). Furthermore, the finite state system approach proposed in this paper can effectively capture the consistency semantic of several historical or longitudinal data.

In the data quality domain accessing the real data is rarely feasible due to economic or practical constraints, indeed the cleansing activities can be performed only relying on domain knowledge. For the same reason this paper focuses on the consistency dimension, while dimensions that require access to the real data (like accuracy) are not considered.

In this section we focused on works that identifies and fix inconsistencies using domain knowledge. We can distinguish among the following paradigms:

Rules based error detection and correction allow users to specify rules and transformation needed to clean the data, a survey can be found in (Maletic and Marcus, 2010). Specifying rules can be a very complex and time consuming task. Furthermore, both bug fixing and maintenance along the time require a non negligible effort.

Several approaches focus on integrity constraints to identify errors, however they cannot address complex errors or several inconsistencies commonly found in real data (Fan, 2008; Maletic and Marcus, 2000). Other constraint types have been identified in the literature: multivalued dependencies, embedded multivalued dependencies, and conditional functional dependencies. Nevertheless, according to Vardi in (Vardi, 1987) there are still semantic constraints that cannot be described by the latter. E.g., the consistency model described in Sec. 6.

Machine learning methods can be used for error localisation and correction. These approaches exploit learning algorithms. After the training an algorithm can be used to identify errors and inconsistencies. Possible techniques and approaches are: unsupervised learning, statistical methods, data profiling, range and threshold checking, pattern recognition, clustering methodologies (Mayfield et al., 2009). The training phase requires a satisfactory dataset to be identified, however a clean dataset that can be used as a reference is rarely available in the data quality field. Therefore, human feedbacks are required to improve the machine learning performances. Since the underlying model built during the learning phase cannot be easily accessed and interpreted by domain experts (e.g., an impact evaluation of the cleansing activities can be hardly done), in this paper we explore a different approach where the consistency models are explicitly stated and verified.

Record linkage (known as *object identification*, *record matching*, *merge-purge problem*) aims to bring together corresponding records from two or more data sources or finding duplicates within the same one. The record linkage problem falls outside the scope of this paper, therefore it is not further investigated. A survey can be found in (Batini and Scannapieco, 2006; Elmagarmid et al., 2007; Maletic and Marcus, 2010).

Consistent query answering works, e.g. (Bertossi, 2006), focus on techniques for finding out *consistent answers* from inconsistent data, i.e. the focus is on automatic query modifications and not on fixing the source data. An answer is considered consistent when it appears in every possible repair of the original database. Semantic constraints are expressed using functional dependencies. The functional dependencies works at the attribute level, therefore they are not well suited to manage consistency issues specific of longitudinal or historical data. Furthermore, already with two Functional Dependencies the problem of computing Consistent Query Answers involving aggregate queries becomes

NP-complete (Bertossi, 2006). In (Bertossi et al., 2011) an approach similar to consistent query answering exploits "matching dependencie"s and "matching functions" instead of functional dependencies. Matching dependencies were introduced as declarative rules for data cleaning and entity resolution. Enforcing a matching dependency on a database instance identifies the values of some attributes for two tuples, provided that the values of some other attributes are sufficiently similar (Bertossi et al., 2011). Matching functions implement the semantic through which different tuples referring to the same entity are made equal. The latter work focuses on data cleansing where mostly record linkage and entity resolution problems are to be addressed. Such problems are not considered in this paper. It is worth to note that the partial order of semantic domination among (cleansed) instances described in (Bertossi et al., 2011), although conceived for a different scenario, can contribute to the process of selecting a correction among a set of several ones i.e., the policy making task briefly introduced in Sec. 5. Due to lack of space, the policy selection process is not further investigated in this paper.

Other works in the field of automata and formal verification theory are now shortly referenced. The application of automata theory for inference purposes was deeply investigated in (Vardi, 1992) in the database domain. The problem of checking (and repairing) several integrity constraint types has been analysed in (Afrati and Kolaitis, 2009). Unfortunately most of the approaches adopted can lead to hard computational problems. Formal verification techniques were applied to databases, to formally prove the termination of triggers (Choi et al., 2006), for semistructured data retrieval (Neven, 2002), and to solve queries on semistructured data (Dovier and Quintarelli, 2009).

Finally, many data cleansing toolkits have been proposed for implementing, filtering, and transforming rules over data. A detailed survey of those tools is outside the scope of the paper. The interested reader can refer to (Maletic and Marcus, 2010).

3 BACKGROUNDS

Model checking (see e.g., (Clarke et al., 1999)) is a hardware/software verification technique to verify the correctness of a given system. The system 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. Focusing on *explicit* model checking techniques, a model checker verifies whether a state transition system always satisfies a property by performing an exhaustive search in the system state-space (i.e., the set of all the feasible system states).

The system is typically modelled as a Finite State System, which can be formally defined 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* $F : S \times A \rightarrow S$ *is the* transition function, *i.e.* $F(s,a) = s'$ *iff the system from state s can reach state* s' *via action a.*

Hence, a trajectory is a sequence of state, action $\pi = s_0 a_0\ s_1 a_1\ s_2 a_2 \ldots s_{n-1} a_{n-1}\ s_n$ *such that* $\forall i \in [0,n], \forall j \in [0,n-1]$, $s_i \in S$ *is a state,* $a_j \in A$ *is an action, and* $F(s_i,a_i) = s_{i+1}$.

Let $\mathcal{S}$ be an FSS according to Def. 1 and let φ be an invariant condition specifying some properties to be satisfied e.g., some consistency properties. Let a state $s_E \in E$ be an error state if the invariant formula φ is not satisfied. Then, the set of *error states* $E \subseteq S$ is defined as the union of the states violating φ. 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.

Informally speaking, a *model checking problem* is composed by a description of the FSS to be explored (by means of a model checker tool language), an invariant to verify and a finite horizon. A feasible solution, or *error trace* (if any) is a trajectory leading the system from an initial state to an error one. Generally speaking, a model checker is usually applied to verify the correctness of a system *model*. In our context, we use a model checker (i) to verify the data consistency (i.e., if the *data* are conform to the model); (ii) to synthesise a set of corrective actions (i.e., all the feasible corrections activities to cleanse the data).

4 DATA CONSISTENCY VIA FSS

Finite State Systems are used to model event-driven systems where the events are mapped to the actions of Def. 1. A bridge between databases (containing longitudinal data) and event-driven system is required to perform data quality verification using model checking techniques. This connection can be done by portraying 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, and an ordered set of records as an *event sequence*. To better clarify this concept, we formalise the following.

Definition 2 (Event, Event Sequence, and Finite State Event Dataset)**.** *Let $\mathcal{R} = (R_1, \ldots, R_n)$ be a schema relation of a database, let $e = (r_1, \ldots, r_m)$ be an* event *where $r_1 \in R_1, \ldots, r_n \in R_n$, then e is a* record *of the projection $(R_1, \ldots, R_m)$ over $\mathcal{R}$ with $m \leq n$.*

A total order *relation $\sim$ on events can be defined such that $e_1 \sim e_2 \sim \ldots \sim e_n$. An* event sequence *is a $\sim$-ordered sequence of events $\varepsilon = e_1, \ldots, e_n$. A* Finite State Event Dataset *(FSED) is an event sequence derived from a longitudinal dataset.*

Intuitively, the application of model checking techniques to data quality problems is driven by the idea that a *model* describing the consistent evolution of *feasible* event sequences (i.e., a *consistency model* expressed by means of FSSs) can be used to verify if the *actual data* (i.e., data retrieved from the database) follow a consistent behaviour. Then, the problem of verifying a database content consistency can be expressed as a model checking problem on FSSs: a, solution for the latter (if any) will represent an inconsistent sequence of tuples for the former. Hence, from here on, we will refer without distinction to an *action* as an *event* and vice versa.

Although a whole database content could be checked by a single FSS, in several domains it is advisable to split the data into different subsets (e.g., for computational reason). Then, the subsets (each being a separate FSED) can be checked separately. To this aim we introduce the following:

Definition 3 (Finite State Event Database)**.** *Let S_i be a FSED, we define a* Finite State Event Database *(FSEDB) as a database DB whose content is $DB = \bigcup_{i=1}^{k} S_i$ where $k \geq 1$.*

It should be clear that performing a model-based data consistency evaluation requires a twofold effort: (1) to define a consistency model of the data evolution, and (2) to verify the data source (e.g., the FSEDB introduced before) against the consistency model. A schematic representation on how this task can be accomplished by using a model checker is depicted in Fig. 1(b). We can distinguish three different phases:

step1 (Data Modelling) A domain expert defines the *consistency model* (e.g., Fig. 1(a)) describing the correct evolution of the data through the model checking tool language;

step2 (Data Verification) A dataset S_i is retrieved from the data source (S). The model checker automatically generates an FSS representing the evolution of the model defined by S_i.

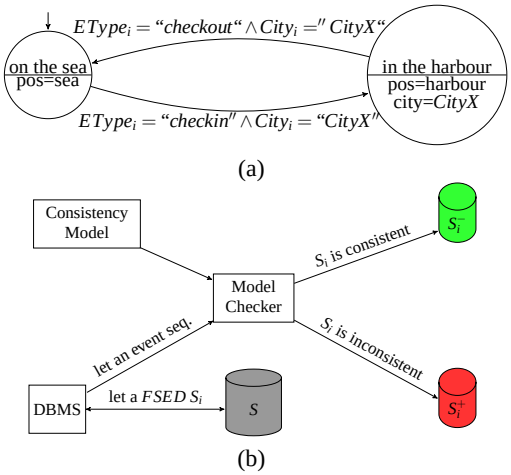


Figure 1: (a) A Graphical representation of the consistency model of the Travel Plan of a Cruise Ship domain. The lower part of a node describes how the system state evolves when an event happens. (b) A Graphical representation of a model checking based data consistency verification of a FSEDB.

step3 (Data Analysis) The model checker looks for an error trace on the FSS. A solution (if any) represents an inconsistency affecting the dataset S_i. Otherwise the event sequence is considered consistent.

The Cruise Ship Example. The following example should clarify the matter. Let us consider the Cruise Ship example as introduced in Tab. 1.

An FSED is the travel plan of a ship, the set of the travel plans of the different ships is the FSEDB. An *event* e_i is composed by the attributes *ShipID*, *City*, *Date*, and *Event Type*, namely $e_i = (ShipID_i, City_i, Date_i, EType_i)$. Moreover, the total-order operator $\sim$ could be the binary operator $\leq$ defined over the event's attribute *Date*, hence $\forall e_i, e_j \in E, e_i \leq e_j$ iff $Date_{e_i} \leq Date_{e_j}$. Finally, a simply consistency property could be *"if a ship checks in to harbour A, then it will check out from A before checking in to the next harbour"*.

We can model this consistency property through an FSS. A graphical representation is given in Fig. 1(a), where the lower part of a node is used to describe how the system state evolves when an event happens. In our settings, the system state is composed by (1) the variable *pos*, which describes the ship's position, and (2) the variable *city* describing the city where the ship has arrived.

The data source S is an actual database instance (e.g., an actual FSEDB) to be verified against the consistency model. In such a case, for each different S_i

(i.e., for each different FSED) the model checker generates a different FSS modelling the S_i consistency evolution.

4.1 From *actual data* to *symbolic data*

Unfortunately, since the consistency verification is strongly related to the actual data (i.e., the FSS expanded by the model checker models the evolution of the database data), the identification of "generic" inconsistent patterns or properties is hard to be accomplished.

To this aim, we use an abstraction of the actual data, namely the *symbolic data* [1], to discover generic inconsistency patterns as well as to identify common data properties. The following example should clarify the concept.

The Cruise Ship Example. Let us consider again the Cruise Ship example of Tab. 1. We recall that $e_i = (ShipID_i,\ City_i,\ Date_i,\ EType_i)$, e_i is is an *event*, and each sequence or subsequence of events is ordered with respect to the date values. Let us consider two inconsistent event sequences, related to two different ships, respectively $S_1 = (checkin, Venice), (checkout, Barcelona)$ and $S_2 = (checkin, Lisbon), (checkout, Naples)$. For the sake of simplicity, we focus on very short sequences. As described before, these event sequences will result in the generation of different FSSs. Nevertheless, the inconsistencies found share a common characteristic: the checkout has been made in a harbour different from the one where the last check-in took place.

We replace the actual city domain data $D_{city} = \{Venice, Barcelona, Lisbon, Naples, \ldots\}$ with a symbolic domain composed by a (small) set of symbols to identify some common inconsistency patterns in the previous example. In other words, we can make an abstraction of the domain D_{city} by using only two symbols, namely $D_{City}^{symbolic} = \{City_X, City_Y\}$. Once a map between actual to symbolic data has been done, we can model the domain as shown in Tab. 2.

The number of symbols to be used, i.e. the symbolic set cardinality has to be chosen according to the criteria described below. More formally, we define the following.

Definition 4 (Symbolic Data and Symbolic Domain)**.** *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 Symbolic Domain *of D as a set of* different *symbols $d_1, \ldots, d_{n'+m'}$, called* Symbolic Data*, required to represent the values of D in the consistency model, i.e. $D^{symbolic} = \{d_1, \ldots, d_{n'+m'}\}$.*

In the Cruise Ship example the *city* state variable and the $City_i$ event attribute both refer to the City domain, therefore the latter can be replaced by the symbolic domain $D_{City}^{symbolic} = \{City_X, City_Y\}$ in the automaton of Fig. 1(a). Finally, some trivial conditions should be met before exploiting a Symbolic Domain rather than the Actual Domain: (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-symbolic value (e.g. $city =$ "$Venice$" in the Cruise Ship example).

Table 2: Values of the domain variables of the Cruise Ship Example.

Variable Type	Variable	Domain Values
State Variables	Pos	sea, harbour
	City	$City_x$, $City_y$
Event data	City	
	Event Type	checkin, checkout

5 DATA CLEANSING VIA FSS

In the previous sections we described how the consistency of a database event sequence can be modelled and verified through model checking. Looking forward, one can wonder if the consistency model can be used as the basis to identify *cleansing activities*. Namely, once the FSS describing the dataset consistency evolution is generated, can the FSS be exploited to identify the corrective events (or actions) able to cleanse an inconsistent dataset?

Let us consider an inconsistent event sequence having an action a_i that leads to an inconsistent state s_j when applied on a (reachable) state s_i . Intuitively, a corrective action sequence represents an alternative route leading the system from state s_i to a state where the action a_i can be applied (without violating the consistency rules). In other words, a *cleansing action sequence* (if any) is a sequence of actions that, starting from s_i, makes the system able to reach a new

[1] the idea is not new and it is inspired by the *abstract interpretation* technique (Clarke et al., 1994).

state on which the action a_i can be applied resulting in a consistent state. In this paper we assume that corrections cannot delete or modify existing data as we are intended to cleanse the data by preserving as much as possible the source dataset.

More formally we can define the following.

Definition 5 (Cleansing Action Sequence)**.** *Let $\mathcal{S} = (S, I, A, F)$ be an FSS, E be the set of errors states (i.e. inconsistent states) and T be the finite horizon. Moreover,*

- *let $\Omega = \bigcup_{i_i \in I}$ Reach(i_i) be the set of all the states reachable from the initial ones;*
- *let $\pi = s_0 a_0 \ldots s_i a_i\ s_j$ be an* inconsistent trajectory*, that is a trajectory where $s_j \in \Omega$ is an inconsistent state (i.e., $s_j \in E$) and $s_0, \ldots, s_i \notin E$.*

*Then, a T-*cleansing action sequence *for the pair (s_i, a_i) is a non-empty sequence of actions $A^c = c_0, \ldots, c_n \in A$, such that exists a trajectory $\pi_c = s_0 a_0 \ldots s_{i-1} a_{i-1}\ s_i c_0\ s_{i+1} \ldots s_{i+n} c_n\ s_k a_i$ on $\mathcal{S}$ with $|A^c| \leq T$, where all the states $s_0, \ldots, s_k$ are consistent.*

In the AI Planning field a *Universal Plan* (Schoppers, 1987) is a set of policies, computed off-line, able to bring the system to the goal from any feasible state (the reader can see (Cimatti et al., 1998; Della Penna et al., 2012) for details). Similarly, we are interested in the synthesis of an object, we call *Universal Cleanser* (UC), which summarises for each *pair* (state, action) leading to an inconsistent state, the set A' of *all* the feasible cleansing action sequences. This UC is computed only once and then applied as an oracle to cleanse any kind of FSEDB. In this sense, a (state, action) pair uniquely represents an *error-code*. To this aim, we proceed as follows:

step1 (Data Modelling) A consistency model of the system is formalised by means of a model checking language as described in Sec. 4.

step 2 (Database Modelling) A *worst-case* FSEDB will be defined, i.e. a fictitious database which contains all the possible event sequences, both the consistent and the inconsistent ones, composed by at most T events for each. Note that this step does not require to really generate such database, indeed it can be easily accomplished by allowing the model to receive any kind of events. For the cruise ship example a worst-case FSEDB is represented by all the possible event sequences $e_1, \ldots, e_T$ where the variable values range in $City_i = \{City_X, City_Y\}$ and $EType_i = \{checkin, checkout\}$. Note that the value of the finite horizon T can be identified as the FSS *diameter*[2].

[2]Due to the limited space we provide only the intuition about how this task can be accomplished. The value is computed by the model checker as the *diameter* of the FSS, i.e. the largest number of states which must be visited in order to travel from one state to another excluding trajectories which backtracks or loops.

step 3 (Data Verification) Use the model checker to generate the FSS representing all the inconsistent sequences, starting from the database domain model (step 2) and the consistency model (step 1), the whole process is shown in Fig. 1(b) as described in Sec. 4

step 4 (UC Synthesis) Explore the FSS to synthesise the Universal Cleanser.

Now we are in state to formalise the Universal Cleansing Problem (UCP) and its solution.

Definition 6 (Universal Cleansing Problem and Solution)**.** *A* Universal Cleansing Problem *(UCP) is a triple $\mathcal{D} = \{\mathcal{S}, E, T\}$ where $\mathcal{S} = (S, I, A, F)$ is an FSS, E be the set of error (or inconsistent) states computed by the model checker, and T is the finite horizon.*

A solution for $\mathcal{D}$, or a Universal Cleanser *for $\mathcal{D}$ is a* map *$\mathcal{K}$ from the set $\Omega \times A$ to a subset A' of the power set of A, namely $A' \subseteq 2^A$, where for each inconsistent trajectory $\pi = s_0 a_0 \ldots s_i a_i\ s_j$ if $A' \neq \emptyset$ then A' must contain* all the possible *T-cleansing action sequences for the pair (s_i, a_i).*

It is worth to highlight that, while on the one hand the UC generated is *domain-dependent*, i.e. it can deal only with event sequences conforming to the model that generated it, on the other hand it is *data-independent* since, once the UC is computed on a worst-case FSEDB, it can be used to cleanse *any* FSEDB. The pseudo code of the algorithm generating a Universal Cleanser is given in Algorithms 1 and 2. It has been implemented on the top of the UPMurphi tool (Della Penna et al., 2009) which has been enhanced with a disk-based algorithm to deal also with big state spaces (Mercorio, 2013). The Algorithm 1 takes as input the FSS specification of the domain, the set of error states given by the model checker (to identify inconsistent trajectories) and a finite horizon T. Then, it looks for a cleansing action sequence (according to Def. 5) for each inconsistent (state, action) pair. This work is recursively accomplished by the Algorithm 2 which explores the FSS through a Depth-First visit collecting and returning all the cleansing solutions.

Running Example. Consider again the Cruise Ship example of Tab. 1. We recall that an *event* e_i is $e_i = (ShipID_i, City_i, Date_i, EType_i)$ and each event sequence and subsequence is ordered with respect to the event dates. It is worth to note that the finite horizon

Table 3: 2-steps Universal Cleanser for the Cruise Ship Example.

([state],(action))	Corrective Actions
([$pos = sea$], ($checkout, City_X$))	($checkin, City_X$)
([$pos = harbour \wedge city = City_X$], ($checkout, City_Y$))	($checkout, City_X$) ($checkin, City_Y$)
([$pos = harbour \wedge city = City_X$], ($checkin, City_Y$))	($checkout, City_X$)
([$pos = harbour \wedge city = City_X$], ($checkin, City_X$))	($checkout, City_X$)

$T = 2$ is enough to guarantee that any kind of inconsistency will be generated and then corrected using no more than 2 actions. Consider that the main elements of an event are $EType_i \in \{checkin, checkout\}, City_i \in \{City_X, City_Y\}$, i.e., 4 possible events. Then, we represent the *wost-case* FSEDB by considering into our model all the possible 2-step event subsequences (i.e., simply enrich each node of the graph in Fig. 1(a) with all the possible edges). Table 3 shows the Universal Cleansing for our example, which is *minimal* with respect to the number of event variable assignments, i.e., the missing pair ([$pos = sea$], ($checkout, City_Y$)) fits on ([$pos = sea$], ($checkout, City_X$)). The UC, once generated, is able to cleanse any kind of $FSEDB$ compliant with the model from which it has been generated.

Algorithm 1 UNIVERSALCLEANSING

Input: $FSS\ \mathcal{S}$,
set of error states E,
finite horizon T
Output: Universal Cleanser $\mathcal{K}$
1: $level \leftarrow 0$; //to stop when T is reached
2: **for all** $s \in S, a \in A$ s.t. $F(s,a) = s^{err}$ **do**
3: $\mathcal{K}[s,a] \leftarrow$ AUXUC$(s,a,level)$
4: **return** $\mathcal{K}$

The Fig. 2(a) describes the overall cleansing process. As a first step, a consistency model of the domain is defined while the Universal Cleanser is automatically synthesised according to the procedures presented in Sec. 4. Then, the "Consistency Verification" task verifies each sequence of the source (and dirty) database *S*. When an inconsistency is found the "Cleanse the Inconsistency" task scans the UC looking for a correction. Since the UC may provide more than one corrective actions for the same inconsistency, a criterion (i.e., a *policy*) is required to select a suitable correction. Once a correction has been properly identified, the inconsistency is fixed and the new sequence is verified iteratively until no further inconsistencies are found. Finally, the new cleansed events sequence is stored into the database "S Cleansed".

It is worth noting that the cleansed results may vary as the policy varies (i.e., the cleansed database

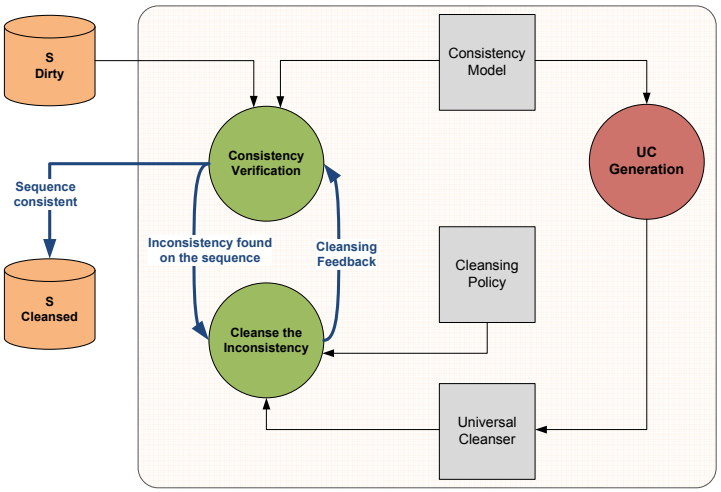
Algorithm 2 AUXUC

Input: a state s,
an action a,
a finite horizon $level$
Output: list of correction sequences $cs[]$
1: $cs[] \leftarrow \emptyset$ //list of correction sequences
2: $cs_{aux}[] \leftarrow \emptyset$ //aux list of correction sequences
3: $i \leftarrow 0$ //local $cs[]$ index
4: **if** $level < T$ **then**
5: **for all** $a' \in A$ s.t. $F(s,a') = s'$ with $s' \notin E$ **do**
6: **if** $F(s',a) = s''$ s.t. $s'' \notin E$ **then**
7: $cs[i] \leftarrow a'$
8: $i \leftarrow i+1$
9: **else**
10: $cs_{aux}[] \leftarrow$ AUXUC$(s',a,level+1)$
11: **for all** $seq \in cs_{aux}$ **do**
12: $cs[i] \leftarrow a' \cup seq$
13: $i \leftarrow i+1$
14: **return** $cs[]$

depends upon the input set of policies) which can be fixed as well as evolve during the cleansing phase, e.g., by using learning algorithms. Clearly, the best suited policy for a given domain can be selected according to several criteria, which often is driven by the data analysis purposes. As an example, one could be interested in studying the variation of an indicator value computed on the cleansed data. To this aim, a policy able to maximise or minimise the value of such indicator should be applied, see e.g. (Mezzanzanica et al., 2012). A discussion on how select a suitable set of policies falls out of the scope of this work. Nevertheless, once the UC has been synthesised any kind of policy can be applied.

6 AN OPEN DATA BENCHMARK PROBLEM

The domain we are presenting is freely inspired by the Italian labour market domain. Indeed, since the 1997, the Italian public administration has been developing an ICT infrastructure, called the *"CO System"*,



(a)

State	emp_k[FT,Limited,CompanyX]
Action	(st,PT,Limited,CompanyY)
Seq. 1	(cs,FT,Limited,CompanyX) → unemp
Seq. 2	(cn,PT,Limited,CompanyX) → emp_i
Seq. 3	(cn,PT,Unlimited,CompanyX) → emp_i
Seq. 4	(cn,FT,Unlimited,CompanyX) (cs,FT,Unlimited,CompanyX) → unemp
...	...
Seq. 9	(cn,FT,Unlimited,CompanyX) (cn,FT,Limited,CompanyX) (cn,PT,Unlimited,CompanyX) → emp_i

(b)

Figure 2: (a) A graphical representation of the Consistency Verification and Cleansing Processes. (b) Some corrective action sequences given by the UC for the error-code 289 with $st = start$, $cs = cessation$, $cn = conversion$ and $ex = extension$.

for recording data concerning employment and active labour market policies, generating an administrative archive useful for studying the labour market dynamics, e.g., (Martini and Mezzanzanica, 2009). According to the Italian labour market laws, 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, managed at local level. These communications are called *Mandatory Communications*.

6.1 Domain Modelling

Each mandatory communication is stored into a record composed by the following attributes:

e_id: it represents an id identifying the communication;

w_id: it represents an id identifying the person involved in the event;

e_date: it is the event occurrence date;

e_type: it describes the type of events occurring to 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: it states whether the event is related to a full-time or a part-time contract;

c_type: describes the contract type with respect to the Italian law. Here we consider *Limited*, i.e. fixed-term, and *unlimited*, i.e. unlimited-term, contracts.

empr_id: it uniquely identifies the employer involved in the event.

The evolution of a consistent worker's career along the time is described by a *sequence* of events ordered with respect to *e_date* and grouped by *w_id*: the sequence can be considered as longitudinal data. Considering the terminology introduced in Def. 1 and Def. 2, an FSED is the ordered set of events for a given *w_id*, and the FSEDs union composes the FSEDB. Now we closely look to the worker careers consistency, where the consistency semantics is derived from the Italian labour law, from the domain knowledge, and from the common practice. Here are reported some constraints:

c1: an employee can have no more than one full-time contract active at the same 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: 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 and fixed-term contract cannot be turned into a full-time contract by an extension;

c4: 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.

The UPMurphi tool allows us to build an FSS unpon which we perform the data consistency task. 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 is showed in Figure 3 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 are handled by the FSS generated by 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., $unemp, emp_i, emp_{i,j}, emp_i, emp_{i,j}$).

From actual to symbolic data. A mapping from actual to symbolic data has been identified as described in Sec 4.1 taking into account both states and events of the automaton of Fig. 3.

We recall that, for the sake of clarity the automaton shows only the consistent transitions triggered by the events allowed in a state, whilst the model checker automatically manages also inconsistent transitions, i.e. transitions triggered by events that lead to an "error state".

The attributes *c_type*, *e_type*, and *c_flag* are already bounded and we left them as is, while the *empr_id* attribute domain has been mapped on a symbolic set of 3 symbols $\{empr_x, empr_y, empr_z\}$ according to the process described in Sec. 4.1.

Finally, we highlight that the model satisfies the conditions p1 and p2 introduces in the Sec. 4.1, 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 a symbolic value with a non symbolic one.

6.2 Experimental Results

Here we comment some results about the consistency verification process performed on the dataset presented in Sec. 6.3. Note that, in order to analyse the quality of the source dataset (wrt consistency), UPMurphi stops the verification algorithm when an inconsistency is found, avoiding the evaluation of the remaining part of the career. Indeed a further evaluation of a career consistency may be affected by the cleansing policy applied, then falsifying the results about the quality of the source dataset.

As first step, we synthesised the UC, identifying 342 different error-codes, i.e. *all* the possible 3-steps (state,action) pairs leading to an inconsistent state of the model. Then, the verification process on the dataset caught $92,598$ inconsistent careers (i.e., the 43% of total careers). The Fig. 4 shows a graphical distribution of the error-codes found. The x-axis reports the error-codes of the UC while the y-axis summarises the number of careers affected by that error. Several analyses can be performed on such consistency outcomes. Nevertheless, since the aim of this work is to provide a technique and a benchmark dataset so that other approaches, comparisons and statistical analysis can be performed on such data, we restrict ourselves to consider the following.

- The closer the error-codes, the similar the error characteristics. We discovered that the three most numerous error codes (i.e., 335, 329 and 319 representing about the 30% of total inconsistencies) arose due to an extension, cessation or conversion event received when the worker was in the *unemployed* status. Hence, cleansing activities for such careers may have a great impact on the quality of the cleansed data.
- Some error-codes require no less than 3 corrective actions to cleanse the data. As an example we report the case of the error-code 53: A worker having two active part-time contracts with *CompanyX* and *CompanyY* receives the cessation of a full-time contract for a third *CompanyZ*. In such a case, a corrective action sequence requires at least three actions to fix the inconsistencies, i.e., to close the contract with *CompanyX* and *CompanyY* and then to start a new full-time contract with *CompanyZ*.
- The UC helps to discover cleansing activities that might otherwise be neglected. To this regard, let us consider the case of a worker having a full-time contract with a *CompanyX* which receives a start of a new part-time contract with *CompanyY*. Looking at the model, a domain expert can argue that probably the worker has closed the full-time contract, but the communication was lost. As a consequence, a hand-written cleansing activity may fix the inconsistency by closing the full-time contract. Nevertheless, for such inconsistency (i.e., the error-code 289) the UC returns 9 different cleansing sequences, as shown in Tab. 2(b), which can contribute in the identification of alternative cleansing policies.

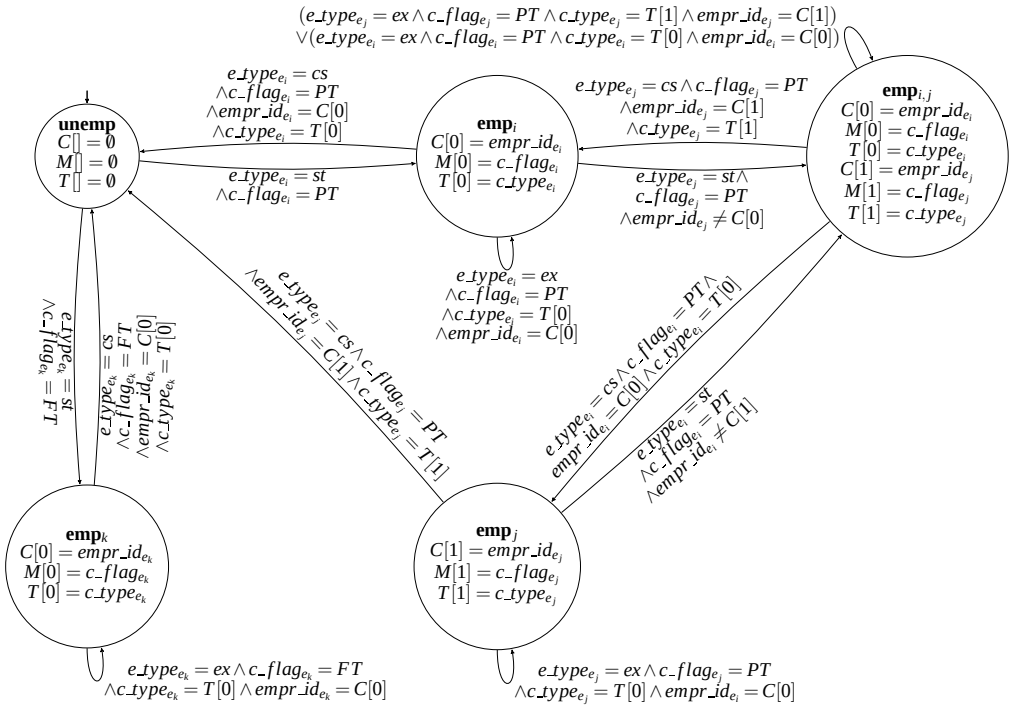


Figure 3: A graphical representation of a valid worker's career FSS where $st = start$, $cs = cessation$, $cn = conversion$ and $ex = extension$.

It is worth noting that applying different corrective actions may lead to different cleansed states, as in the latter example where fixing the inconsistency through Seq.1 leads the worker career to the unemployed state whilst the application of Seq.3 brings the career to a different one (i.e., the *emp_i*). Hence, for a high-quality cleansing process the joint utilisation of UC and domain-dependent policies is required.

6.3 Online Dataset Description

The whole dataset and the experimental results presented in Sec. 6.2 has been made publicly available for download[3]. The source archive contains $1,248,814$ mandatory communications describing the careers of $214,429$ people observed starting from the 1^{st} January 2001 to the 31^{st} December 2010. 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.

The Universal Cleanser. It has been generated according to Def. 6 on the consistency model of Fig. 3.

7 CONCLUDING REMARKS

In this paper we have shown how a model-based approach can be used to verify and cleanse a dirty dataset, providing an algorithm (build on top of the UPMurphi tool) to automatically synthesise a universal cleanser that, as a characteristic, is *domain-dependent* (i.e., it copes with consistency issues for a given domain) but *data-independent* (i.e., it can cleanse any kind of dataset compliant with the model).

Moreover, we presented a real-world scenario in the labour market domain for which the universal cleanser has been computed. As a further contribution, an anonymous version of the dataset used

[3] `http://goo.gl/zrbrR`. The username is: data2013materials@gmail.com Password: data2013

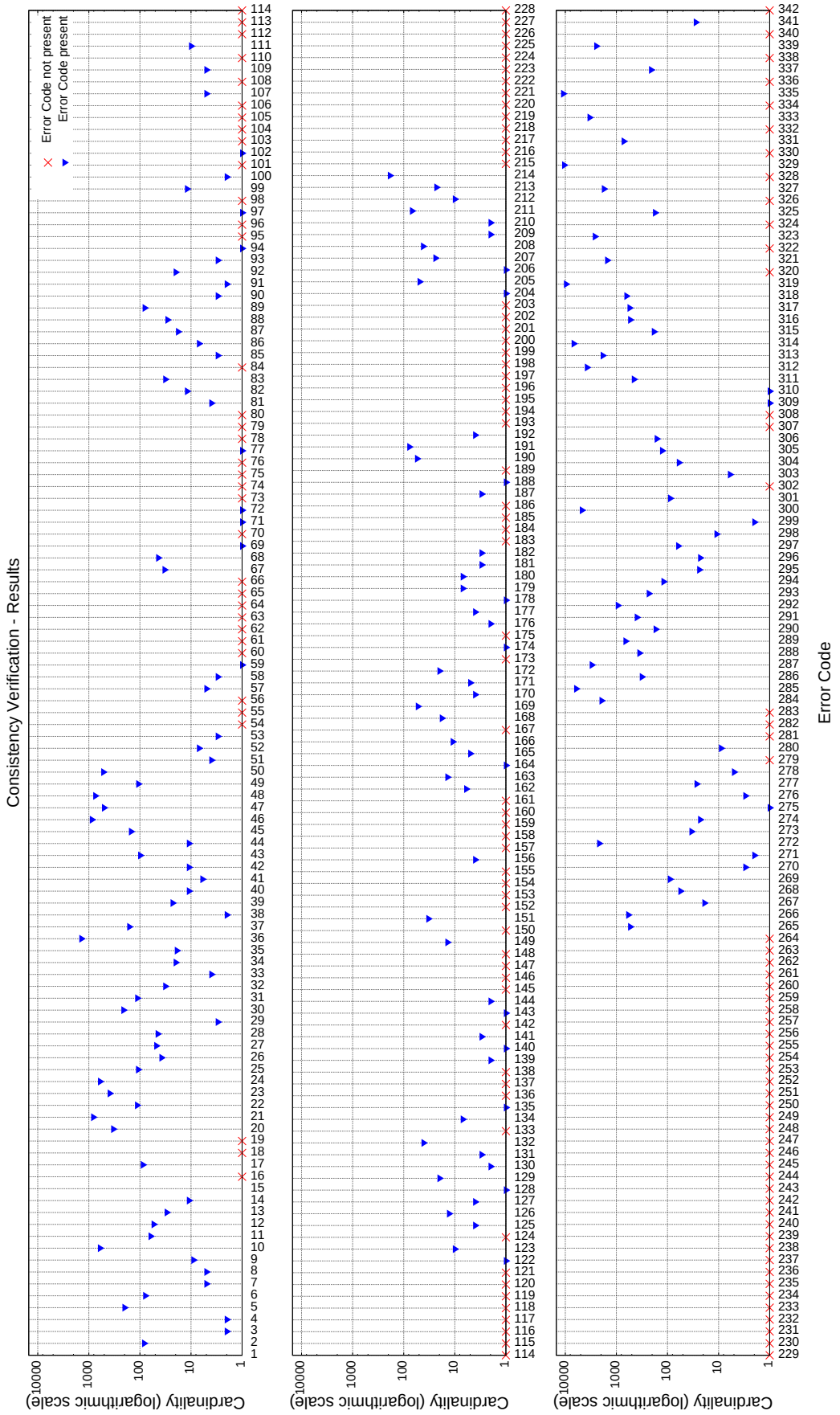


Figure 4: A graphical visualisation of the distribution of the error-codes found.

has been made available for download (according to the current law and privacy requirements) together with the cleanser and the consistency verification outcomes. Our results confirm the usefulness of exploiting model-based verification and cleansing approaches in the data quality field, as it may help domain experts and decision makers to have a better comprehension of the domain aspects, of the data peculiarities, and of the cleansing issues.

ACKNOWLEDGEMENTS

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

REFERENCES

Afrati, F. N. and Kolaitis, P. G. (2009). Repair checking in inconsistent Databases: Algorithms and Complexity. In *ICDT*, pages 31–41. ACM.

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

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

Bertossi, L. (2006). Consistent query answering in databases. *ACM Sigmod Record*, 35(2):68–76.

Bertossi, L. E., Kolahi, S., and Lakshmanan, L. V. S. (2011). Data cleaning and query answering with matching dependencies and matching functions. In Milo, T., editor, *ICDT*, pages 268–279. ACM.

Choi, E.-H., Tsuchiya, T., and Kikuno, T. (2006). Model checking active database rules under various rule processing strategies. *IPSJ Digital Courier*, 2(0):826–839.

Cimatti, R., Roveri, M., and Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *AAAI-98, pp. 875–881.*, pages 875–881. AAAI Press.

Clarke, E. M., Grumberg, O., and Long, D. E. (1994). Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542.

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

Della Penna, G., Intrigila, B., Magazzeni, D., and Mercorio, F. (2009). UPMurphi: a tool for universal planning on PDDL+ problems. In *ICAPS 2009*, pages 106–113. AAAI Press.

Della Penna, G., Magazzeni, D., and Mercorio, F. (2012). A universal planning system for hybrid domains. *Applied Intelligence*, 36(4):932–959.

Dovier, A. and Quintarelli, E. (2009). Applying Model-checking to solve Queries on semistructured Data. *Computer Languages, Systems & Structures*, 35(2):143 – 172.

Elmagarmid, A., Ipeirotis, P., and Verykios, V. (2007). Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16.

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

Fayyad, U. M., Piatetsky-Shapiro, G., and Uthurusamy, R. (2003). Summary from the kdd-03 panel: data mining: the next 10 years. *ACM SIGKDD Explorations Newsletter*, 5(2):191–196.

Maletic, J. and Marcus, A. (2000). Data cleansing: beyond Integrity Analysis. In *IQ*, pages 200–209.

Maletic, J. and Marcus, A. (2010). Data cleansing: A prelude to knowledge discovery. In *Data Mining and Knowledge Discovery Handbook*, pages 19–32. Springer US.

Martini, M. and Mezzanzanica, M. (2009). The Federal Observatory of the Labour Market in Lombardy: Models and Methods for the Costruction of a Statistical Information System for Data Analysis. In *Information Systems for Regional Labour Market Monitoring - State of the Art and Prospectives*. Rainer Hampp Verlag.

Mayfield, C., Neville, J., and Prabhakar, S. (2009). A Statistical Method for Integrated Data Cleaning and Imputation. Technical Report CSD TR-09-008, Purdue University.

Mercorio, F. (2013). Model checking for universal planning in deterministic and non-deterministic domains. *AI Communications*, 26(2).

Mezzanzanica, M., Boselli, R., Cesarini, M., and Mercorio, F. (2012). Data quality sensitivity analysis on aggregate indicators. In *DATA 2012*, pages 97–108. SciTePress.

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

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

Singer, J. and Willett, J. (2003). *Applied longitudinal data analysis: Modeling change and event occurrence*. Oxford University Press, USA.

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

Vardi, M. Y. (1992). Automata Theory for Database Theoreticians. In *Theoretical Studies in Computer Science*, pages 153–180. Academic Press Professional, Inc.