A Model-Based Approach for Developing Data Cleansing Solutions

MARIO MEZZANZANICA, ROBERTO BOSELLI, MIRKO CESARINI, and FABIO MERCORIO, Department of Statistics and Quantitative Methods, C.R.I.S.P. Research Centre, University of Milano-Bicocca, Italy

The data extracted from electronic archives is a valuable asset; however, the issue of the (poor) data quality should be addressed before performing data analysis and decision-making activities. Poor data quality is frequently cleansed using business rules derived from domain knowledge. Unfortunately, the process of designing and implementing cleansing activities based on business rules requires a relevant effort. In this article, we illustrate a model-based approach useful to perform inconsistency identification and corrective interventions, thus simplifying the process of developing cleansing activities. The article shows how the cleansing activities required to perform a sensitivity analysis can be easily developed using the proposed model-based approach. The sensitivity analysis provides insights on how the cleansing activities can affect the results of indicators computation. The approach has been successfully used on a database describing the working histories of an Italian area population. A model formalizing how data should evolve over time (i.e., a data consistency model) in such domain was created (by means of formal methods) and used to perform the cleansing and sensitivity analysis activities.

Categories and Subject Descriptors: H.2.m [**Database Management**]: Miscellaneous; D.2.4 [**D.2.4 Software/Program Verification**]: Formal methods; Model checking

General Terms: Algorithms, Verification, Theory

Additional Key Words and Phrases: Data quality, data consistency, data verification, ETL, data believability

ACM Reference Format:
Mario Mezzanzanica, Roberto Boselli, Mirko Cesarini, and Fabio Mercorio. 2015. A model-based approach for developing data cleansing solutions. ACM J. Data Inf. Qual. 5, 4, Article 13 (February 2015), 28 pages.
DOI: http://dx.doi.org.10.1145/2641575

1. INTRODUCTION

In the last decade, information systems usage has grown apace, fostered by the availability of several Information Communication Technology (ICT)-based services. Hence, a lot of data have been collected from business, social, and governmental transactions, which can deeply describe the ongoing relations among people, public institutions, and organizations. Such data could be used to accurately analyse social, economic, and business phenomena, and to assess decision-making activities like the evaluation of active policies, the allocation of resources, and the design and improvement of (public) services.

A preliminary version of this work appears in M. Mezzanzanica, R. Boselli, M. Cesarini, and F. Mercorio, "Data Quality through Model Checking Techniques," in Proceedings of Intelligent Data Analysis (IDA), Lecture Notes in Computer Science, Vol. 7014, 2011, pp. 270–281.

Authors' addresses: M. Mezzanzanica, R. Boselli, M. Cesarini, and F. Mercorio, University of Milano-Bicocca, Department of Statistics and Quantitative Methods. Via Bicocca degli Arcimboldi 8, II floor. I-20126 Milano, Italy; emails: {mario.mezzanzanica, roberto.boselli, mirko.cesarini, fabio.mercorio}@unimib.it.

DOI: http://dx.doi.org.10.1145/2641575

Table I. Travel Plan of a Cruise Ship

ShipID	City	Date	Event Type
S01	Venice	12th April 2011	check-in
S01	Venice	15th April 2011	check-out
S01	Lisbon	30th April 2011	check-in
S01	Barcelona	5th May 2011	check-in
S01	Barcelona	8th May 2011	check-out
...	...	...	...

Unfortunately, several studies report that the *data quality* of enterprise databases and public administration archives is very poor (e.g., Strong et al. [1997], Redman [1998], and Batini and Scannapieco [2006]); indeed, the consequences on decision making can be very unpleasant unless the quality issues are appropriately addressed. To give an example, the causes of the Challenger Space Shuttle explosion are attributed to 10 different categories of data quality problems [Fisher and Kingma 2001]. Organisations are becoming aware of poor data quality consequences and costs, therefore a lot of effort has been spent to address such issues, as described in Tee et al. [2007].

Data quality is a broad concept encompassing several dimensions (e.g., *accuracy*, *consistency*, and *accessibility*). A complete survey can be found in Batini and Scannapieco [2006]. This article focuses on data cleansing based on a consistency model to support sensitivity analysis. The sensitivity analysis was performed on *longitudinal data*. *Consistency* is a quality dimension paying attention to the violation of semantic rules defined over a set of data items. *Longitudinal data* (also known as *panel data* or *historical data*) refer to a set of repeated observations of the same object or subject at multiple time points (see Hedeker and Gibbons [2006], Singer and Willett [2003], and Bartolucci et al. [2013] for details).

As an example of inconsistent longitudinal data, let us consider the dataset in Table I showing a cruise ship travel plan. The ship usually travels by sea and calls at different ports (intermediate destinations) doing a *check-in* when entering and a *check-out* when exiting from the harbor. The reader will notice that the ship cannot enter in Barcelona unless it has checked out from Lisbon. In this respect, the dataset is inconsistent.

Missing data and wrong values (just to cite a few poor quality issues) may pass unnoticed during the "daily operations," but they can strongly affect the information derived for decision-making purposes. Therefore, the implementation of quality improvement activities is strongly required for analytical purposes. When no different (and more trusted) data source is available for comparing the data, a feasible solution is to perform data cleansing based on business rules, that is, to implement cleansing algorithms that try fixing inconsistencies using domain-derived knowledge.

The work described in this article is driven by the idea that a *model* describing the consistent evolution of longitudinal data can be used to identify inconsistencies and to perform cleansing activities.

The next sections are organized as follows: Section 2 will describe this article's contribution and the related work; Section 3 will outline how formal methods (specifically model checking) can be used to perform data consistency verification; Section 4 will show how the model-based approach can be used to execute cleansing activities; Section 5 will describe a real case where the consistency checking and cleansing approach described in this article have been used; the conclusion and the future works are outlined in Section 6.

2. CONTRIBUTION AND RELATED WORK

This article focuses on cleansing activities where the corrective interventions are developed using a model-based approach. A sensitivity analysis is built upon the model-based cleansing and used to assess a dirty dataset containing longitudinal data.

Longitudinal data (aka *panel* or *historical data*)[1] have received a lot of attention from many research communities since such data are used in many real-world instances, including labor and healthcare domains (see, e.g., Hansen and Järvelin [2005], Holzinger [2012], Wong et al. [2011], Boselli et al. [2014a], Holzinger and Zupan [2013], Prinzie and Van den Poel [2011], Boselli et al. [2014b], Lovaglio and Mezzanzanica [2013], Devaraj and Kohli [2000], and Boselli et al. [2014c]).

The data quality analysis and improvement tasks have been the focus of a large body of research that involves 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].

Data quality is a domain-dependent concept, usually defined as "fitness for use," thus quality considered appropriate for one use may not be suitable for another use. Here we shall focus on *consistency*, which considers "the violation of semantic rules defined over (a set of) data items, where items can be tuples of relational tables or records in a file" [Batini and Scannapieco 2006]. Focusing on relational models, such "semantic rules" have usually been expressed through Functional Dependencies (FDs), Conditional Functional Dependencies (CFDs), Denial Constraints (DCs), join dependencies, and inclusion dependencies, useful for specifying integrity constraints. The interested reader can refer to Fan [2008], Bohannon et al. [2007], Papotti et al. [2013], and Chu et al. [2013]. Indeed, as argued in Chomicki [1995], FDs are expressive enough to model static constraints, which evaluate the current state of the database, but they do not take into account the past (i.e., how the database has evolved over time). Furthermore, even though FDs enable the detection of errors, they have limited usefulness since they fall short of acting as a guide in correcting them [Fan et al. 2010]. Finally, FDs are only a fragment of first-order logic and this motivates the usefulness of formal systems in databases, as studied by Vardi [1985]. Furthermore, DCS are not well suited for addressing the inconsistency problems described in this article, as outlined in Appendix A.

The longitudinal data expected behavior (i.e., how the data is expected to evolve over time) can be modeled using graphs or tree formalisms as in the case of *weakly structured* data (see, e.g., Holzinger [2012]). Specifically, let $Y(t)$ be an ordered sequence of observed data (e.g., subject data sampled at different time $t \in T$); the observed data $Y(t)$ can be represented by a path on a graph. In this regard, model checking formalisms are well suited to model such complex constraints. Intuitively, a model checking domain would describe how data (recorded in a database and representing the interaction between a subject and the external word) may change the subject status,[2] while a model checking instance would be initiated with actual data to evaluate (as a problem goal) if the data are compliant to the domain model. Model checking can be also used to identify corrective actions to fix data inconsistencies, where the corrective actions are automatically inferred from the domain model.

The proposed approach allows domain experts to concentrate on *modeling* the quality constraints rather than on *how* to verify and enforce them. The approach proposed in this article focuses on model checking to build a single holistic consistency model, since model checking supports undesired behavior prevention and properties verification

[1]*Longitudinal data* extracted by ISs provide knowledge about a given subject, object, or phenomena observed at multiple sampled time points, as introduced in Section 1.
[2]"Status" here is considered in terms of a value assignment to a set of finite-domain state variables.

(e.g., identifying constraints conflicting each other). This helps reduce the effort of creating cleansing procedures, which is a complex and error-prone activity.

From an academic point of view, there is a large amount of literature exploiting dependency theory (i.e., integrity constraints) for specifying data consistency and performing data repair. Two widely investigated approaches based on FDs are *database repair* and *consistent query answering* [Chomicki and Marcinkowski 2004]. The former aims at finding a repair, that is, a database instance that satisfies integrity constraints and minimally differs from the original one, while the latter 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 becomes NP-complete already using two (or more) FDs, as observed in Bertossi [2006]. To mitigate this problem, a number of works have recently exploited heuristics for finding a database repair [Yakout et al. 2013; Kolahi and Lakshmanan 2009]. These approaches seem to be very promising, even though their effectiveness has not been evaluated on real-life domains.

From an industry perspective, a lot of off-the-shelf tools are available and well supported, but they often lack formality in addressing domain-independent problems, as is the case of several ETL tools.[3] A gap between practice-oriented approaches and academic research contributions still exists in the data quality field, as reported in Batini et al. [2009]. For example, the declarative constraints studied in the research community [Papotti et al. 2013] could greatly contribute to the cleansing development processes, since the procedural business rules currently used in the enterprise settings require one to perform manually or by ad hoc routines a quite relevant amount of data analysis and cleansing work. Furthermore, the developed routines may be difficult to write and maintain, as discussed in Rahm and Do [2000].

Recently, the NADEEF [Dallachiesa et al. 2013] and LLUNATIC [Geerts et al. 2013] tools were developed to unify the most used cleansing solutions by both academy and industry through variants of FDs. Specifically, NADEEF provides a programming interface facilitating the user in expressing quality constraints through business rules, conditional functional dependencies, and matching dependencies. 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 (1) a single tuple, (2) two tuples, or (3) 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 is the case of longitudinal data (this is supported also by the example provided in Appendix A), and this makes the exploration-based technique (as the model checking is) a good candidate for that task. Formal verification techniques were used in the database domain to investigate the termination of triggers [Choi et al. 2006; Ray and Ray 2001].

Record linkage (known as *object identification*, *record matching*, *merge-purge problem*) aims to bring together corresponding records from two or more data sources or find duplicates within the same one. The record linkage problem falls outside the scope of this article, therefore it is not further investigated. The interested reader can refer to Elmagarmid et al. [2007] and Yakout et al. [2012].

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

The contribution of this article is twofold:

(1) to illustrate a model-based approach for developing data cleansing solutions, which simplifies the development of cleansing activities, and hence different ways of cleansing a dataset can be easily created and evaluated;
(2) to propose a methodology to quantitatively estimate the impact that uncertainty (due to low data quality) can have on indicators computed on cleansed data.

Furthermore, the proposed methodologies have been implemented and validated against a real-world dataset extracted by a public administration information system.

3. MODEL-BASED DATA CONSISTENCY VERIFICATION

A bridge between databases containing longitudinal data and *event-driven systems* is introduced. A database record can be interpreted as the description of an event (that can modify a *system*), and an ordered set of records can be viewed as an *event sequence*. Consequently, the system verification techniques commonly used in event-driven systems can be used also in the data domain.

To better clarify the concepts, we formalize the following.

Definition 3.1 (*Event, Event Sequence, and Finite State Event Dataset*). Let $\mathcal{R} = (R_1, \ldots, R_l)$ be a schema of a database relation. Then,

(i) an *event* $e = (r_1, \ldots, r_m)$ is a *record* of the projection $\mathcal{Q} = (R_1, \ldots, R_m)$, where $\mathcal{Q} \subseteq \mathcal{R}$ with $m \leq l$ such that $r_1 \in R_1, \ldots, r_m \in R_m$ (i.e., only a subset of the $\mathcal{R}$ attributes are considered);

(ii) let $\sim$ be a *total order* relation over events, an *event sequence* is a $\sim$-ordered sequence of events $\epsilon = e_1, \ldots, e_k$ concerning the same object or subject;

(iii) a *Finite State Event Dataset* (FSE-D) is an event sequence derived from a longitudinal dataset.

In several domains, it is advisable to split a (large) dataset into different subsets (e.g., for computational reasons). Then, each subset can be managed separately (e.g., parallel computation can be performed). From now on, the term *FSE-D* will be used to refer to a subset, while the overall dataset will be called *FSE-DB*.

Definition 3.2 (*Finite State Event Database*). Let S_i be an FSE-D; we define the *Finite State Event Database* (FSE-DB) as a dataset *DB* whose content is $DB = \bigcup_{i=1}^{k} S_i$, where $k \geq 1$.

By abuse of notation, the term FSE-DB will also be used to refer to a data source (e.g., a database), whose content can be partitioned in several FSE-Ds for analysis purposes.

The following example should clarify the matter. Let us consider the cruise ship example, as introduced in Table I.

An FSE-D is the travel plan of a ship, and the set of the (different ships) travel plans is the FSE-DB. An *event* e_i is a record that can be broken down into 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 $\forall e_i, e_j \in E, e_i \leq e_j$ if and only if $Date_{e_i} \leq Date_{e_j}$.

Some model checking backgrounds are introduced before outlining how model checking can be used to perform data consistency verification.

3.1. Model Checking Backgrounds

Model checking (see, e.g., Clarke et al. [1999] and Baier and Katoen [2008]) is a hardware/software verification technique to verify the correctness of a system. Focusing on *explicit* model checking, both the system model and the desired properties

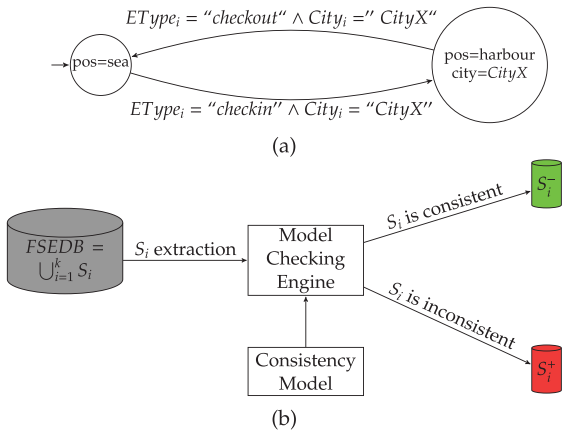

Fig. 1. (a) A graphical representation of the FSS (a consistency model) that can be used to verify the consistency of the travel plan of a cruise ship. The content of every node describes how the state evolves when an event happens (i.e., what are the values assigned to the state variables). (b) A graphical representation of a model checking based data consistency verification on an FSE-DB. Each S_i is extracted from the FSE-DB and analyzed by the model checker 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^+), according to the results of the model checking task.

can be described by means of a model checker language. Then, the model checker generates a corresponding Finite State System (FSS) from the model checker language description where the desired properties can be evaluated.

Some concepts are briefly summarized. A FSS can be formally defined as follows.

Definition 3.3 (*FSS*). A *FSS* $\mathcal{S}$ is a 4-tuple (S, I, A, F), where S is a finite set of *states*,[4] $I \subseteq S$ 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*, that is, $F(s, a) = s'$ if and only if the system from state s can reach state s' via action a.

A trajectory compliant with the FSS is a sequence of *state, action* $\pi = s_0a_0\ s_1a_1\ s_2a_2 \ldots s_{n-1}a_{n-1}\ s_n$ such that $\forall i \in [0, n-1]$, $s_i, s_{i+1} \in S$ are states, $a_i \in A$ is an action, and $F(s_i, a_i) = s_{i+1}$.

Since a system can evolve in different ways, several trajectories can be generated. The set of possible trajectories can be partitioned according to specific classification criteria (e.g., whether a trajectory meets a specific criteria or not).

Let $\mathcal{S}$ be an FSS according to Definition 3.3 and let φ be some properties to be satisfied (called *safety properties* or *invariant* in the model checking domain). 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 where φ is violated. The error states and the related transitions are not shown in the FSS in Figure 1(a) for the sake of simplicity. The reader can figure out that a *check-in* triggered transition from the *in the harbor* state (the state where pos=harbor) will lead the system to an error state. An invariant formula example for the FSS in Figure 1(a) can require the system never entering into an error state.

We limit the error exploration (in an FSS) to at most T actions (the *finite horizon*), that is, only sequences reaching an error $s_E \in E$ within the finite horizon are detected.

[4]In an FSS, a state is a configuration of the state variables (i.e., a set of values assigned to the state variables); the interested reader can refer to Clarke et al. [1999] and Lee and Yannakakis [1996].

Note that this restriction has a limited impact in our contexts,[5] although being theoretically quite relevant; the interested reader can refer to Clarke et al. [2003] and Biere et al. [2003].

Informally speaking, a *model checking problem* is composed by a description of the FSS to be explored, 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. In a typical model checking problem, the FSS and the invariant are not described directly, but they are derived by the model checker from a problem specification described using the model checker language. In this way, problems can be described in a concise yet comprehensive way.

For the sake of completeness, we remark that the high formalization and computational power of the model-checking paradigm has been applied to several contexts far from the HW/SW verification, from the control theory to the AI planning (see, e.g., Giunchiglia and Traverso [2000] and Henzinger et al. [1997]). As a consequence, several model checking tools are available; in our context, we used the CMurphi verifier [CMurphi Web Page 2011], which proved its effectiveness as the core of several tools, from the automated controller generation [Magazzeni 2011; Della Penna et al. 2008] to planning in both deterministic and nondeterministic domains [Della Penna et al. 2009, 2011].

3.2. Data Verification

The data consistency verification of the cruise ship data introduced in Table I can be performed using a FSS like the one described in Figure 1(a).

An event, as defined in Definition 3.1, can be viewed as the description (or the transcription) of an action that affects an FSS. For this reason, from now on the terms event, action, and (database) record (containing longitudinal data) will be used interchangeably.

The upper part of a node shows the state name, and the lower part 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 reader can see that the FSS described in Figure 1(a) can be used to detect the inconsistency previously found in Table I: A transition triggered by the *Lisbon check-in* event is not allowed from the *in the harbor* FSS state.

The rules that can check the data consistency in the context of a specific domain will be called *data consistency model* hereafter. Once a data consistency model has been identified for a specific domain, the consistency verification of an FSE-D can be expressed as a *model checking* problem. A schematic representation on how a model checker can identify data inconsistencies is depicted in Figure 1(b).

Generally speaking, a model checker verifies if a model can represent a system or not. In this article, the actual data play the role of the system behavior description, and the consistency model is the inquired model. In other words, the model checker investigates whether the consistency model represents the actual data or not. Since a model checker can find out a portion of a data sequence invalidating the model, it can be used as an inconsistency hunter.

By using a model checker, the effort of implementing a data consistency verification process focuses on the consistency model design, thus greatly simplifying the overall process. For this reason, this approach has been called model-based consistency

[5]Let us suppose that for every ship of the cruise ship example described in Section 1, the number of *check-in* and *check-out* operations does not exceed a value Z, then a finite horizon $T = Z$ is fine to check all the ship trajectories.

Table II. The SDD for the Cruise Ship Example

Variable Type	Variable	Domain Values
State variables	Pos	Sea, harbor
	City	$City_x, City_y$
Event data	City	
	Event type	Check-in, check-out

verification. The examples described in the next sections will focus on longitudinal data. Nevertheless, the proposed approach can also be used for different data types, whereas a consistency semantic can be modeled over the data sequences.

3.3. From *Actual Data* to *Symbolic Data*

In the data consistency verification task previously described, the system behavior description is replaced by the actual data. As a consequence, the identification of "generic" inconsistency patterns, properties, and cleansing interventions is hard to accomplish. The following example should clarify the concept.

Let us consider again the cruise ship example of Table I. We recall that $e_i = (ShipID_i, City_i, Date_i, EType_i)$ is an *event*, and each event sequence (or subsequence) is ordered with respect to the date values. Let us consider the data describing two ship journeys, respectively, $S_1 = \{(Ship_1, Venice, 9^{th}\ June\ 2011, checkin), (Ship_1, Barcelona, 18^{th}\ July\ 2011, checkout)\}$ and $S_2 = \{(Ship_2, Lisbon, 15^{th}\ July\ 2011, checkin), (Ship_2, Naples, 23^{rd}\ August\ 2011, checkout)\}$. The model checker will generate and evaluate two different FSSs, even if the consistency model is the same. The inconsistencies identified by the model checker in each sequence share a common characteristic: the checkout was performed in a harbor different from the one where the last check-in took place.

The symbolic data are introduced as follows: To identify *generic inconsistencies* (in this scenario), we replace the actual city domain data $D_{city} = \{Venice, Barcelona, Lisbon, Naples, \ldots\}$ with a (small) symbolic set. In other words, we can make an abstraction of the domain D_{city} by using only two symbols, namely, $D^{symbolic}_{City} = \{City_X, City_Y\}$. Then, we can model the domain as shown in Table II. How is the cardinality of the symbolic set chosen? The criteria for identifying how many symbols should be used in a scenario are described more formally in the following.

Definition 3.4 (*Symbolic Data and Symbolic Domain*). Let s be a (generic) state of a consistency model and e be a (generic) event with, respectively, $x_1, \ldots, x_n$ state variables and $e = (r_1, \ldots, r_m)$ event attributes.

Let D_x be an infinite (or a finite but very large) attribute domain where a subset of both the state variables and the event attributes belongs to D_x, (i.e., $\{x_1, \ldots, x_{n'}\} \in D_x$ and $\{r_1, \ldots, r_{m'}\} \in D_x$, and $\{x_1, \ldots, x_{n'}\} \subseteq \{x_1, \ldots, x_n\}$ and $\{r_1, \ldots, r_{m'}\} \subseteq \{r_1, \ldots, r_m\}$).

An event e happening in a state s requires the evaluation of $x_1, \ldots, x_{n'}$ and $r_1, \ldots, r_{m'}$ values of D_x, namely, a configuration of $n' + m'$ values of D_x. Then, we define the *symbolic domain* of D_x as a set of *different* symbols $d_1, \ldots, d_{n'+m'}$, called *symbolic data* (or *symbolic set*), that can represent $n' + m'$ different values of D_x in the consistency model, (i.e., $D^{symbolic}_x = \{d_1, \ldots, d_{n'+m'}\}$).

In the cruse 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 $D^{symbolic}_{City} = \{City_X, City_Y\}$ in the FSS of Figure 1(a).

Finally, some trivial conditions should be met before replacing an actual data domain with a symbolic one:

Table III. Universal Checker for the Cruise Ship Example
The error code 0 is reserved for consistent sequences, s is for the ShipID whose event sequence is being evaluated, and $*$ is for any value.

Error Code	State	Event
		(ShipID, City, Date, Event Type)
1	$pos = sea$	$(s, City_X, *, check\text{-}out)$
2	$pos = harbor \wedge city = City_X$	$(s, City_Y, *, check\text{-}out)$
3	$pos = harbor \wedge city = City_X$	$(s, City_Y, *, check\text{-}in)$
4	$pos = harbor \wedge city = City_X$	$(s, City_X, *, check\text{-}in)$

(1) no total order relation is defined on the actual data domain (or the total order relation is not considered for the scope of the analysis);
(2) no condition should compare a symbol to a nonsymbolic value (e.g., *city* = "*Venice*" in the cruise ship example).

The *symbolic data* are useful to manage attributes having infinite domains or very high cardinalities. A domain of interest where some attributes are described by symbolic data and some others are described by actual data will be called Symbolic Data Description (SDD) hereafter.

3.4. Universal Checker

The consistency framework described in this section can use the SDD and the consistency model to explore every state and event combination leading to an inconsistency. This information is used to build the *universal checker*, a domain-dependent but data-independent set of rules (focusing on state transition system dynamics) that can identify the data not satisfying the consistency model. An example is shown in Table III. The expression universal checker is inspired by the *artificial intelligence planning* terminology, as described in Section 4.

More formally, we define the universal checker as follows (we recall that the terms event and action are used interchangeably as outlined in Section 3.2).

Definition 3.5 (*Universal Checker*). Let $\mathcal{S} = (S, I, A, F)$ be an FSS according to Definition 3.3, and let *Err* be a set of error codes, where $Err \subset \mathbb{N} \cup \{0\}$ and 0 is used for "no error." A *Universal Checker* (UC) is a *map* $\mathcal{K}$ from the set $S \times A$ to the set of error codes Err.

Notice that the symmetry reduction technique was used to identify symmetrical errors: for example, the *state event* pairs $([pos = harbor \wedge city = City_X], (check\text{-}in, City_X))$, and $([pos = harbor \wedge city = City_Y], (check{-}in, City_Y))$ are identified as similar and the second one is dropped. The interested reader can refer to Norris Ip and Dill [1996] for more information on symmetry reduction techniques. Furthermore, an *error code* can be allocated to each UC entry and used for classification and cleansing purposes. The error code univocally identifies a set of similar errors.

The UC can be generated offline for a specific domain of interest (i.e., the model checker does not need accessing the actual data) and then used on the actual data to identify inconsistencies.

3.5. Reset Events

The missing Lisbon departure of Table I prevents the exploitation of the UC straight after the Lisbon check-in. The available information is not enough to guess the state of the FSS implementing the consistency model. Therefore, the uncertainty originating from an inconsistency can prevent the execution of the consistency check for the subsequent events.

Nonetheless, this uncertainty can be narrowed. Considering again the example of Table I, the Barcelona port check-in is enough to guess the FSS state and to resume the checking activities. In other cases (e.g., the one presented in Section 5), the uncertainty can last for some of the subsequent events.

Intuitively, the *reset events* identify points where the consistency check can be safely resumed. A *reset event* leads the FSS always to the same state, independently of the previous history; the FSS state can be determined with certainty thereafter.

More formally, we have the following.

Definition 3.6 (*Reset Action*). Let $\mathcal{S}$ (S, I, A, F) be a *FSS* according to Definition 3.3; an action $a \in A$ is a reset action if and only if $\exists s_a \in S$ such that $\forall s \in S$ either $F(s, a) = s_a$ or $F(s, a)$ is not defined. Since the terms event and action are used interchangeably in this article, as outlined in Section 3.2, by abuse of notation we will use the expression *reset event* to refer to an event that causes a reset action in the FSS considered.

The reset events can be used for partitioning a dataset into smaller event segments that can be independently checked. In this way, an inconsistency does not prevent the evaluation of the subsequent segments.

4. MODEL-BASED DATA CLEANSING

Looking forward, one can wonder if the consistency framework previously introduced can be expanded to perform *cleansing activities*. In the following we investigate such topic.

Let us consider a consistency model, like the one shown in Figure 1(a), and an inconsistent event sequence where an event e_i leads the system from a (reachable) state s_i to an inconsistent state s_j.

A *cleansing event sequence* (if any) is a sequence of events that, starting from s_i, makes the system able to reach a new state on which the event e_i can be applied.

An event like the e_i in the previous example will be called the Consistency Failure Point (CFP). The CFP is not necessarily responsible for the consistency failure, but it is the point where the failure emerges.

This work relies on a precondition: the correction activities focus on adding events and no cancellation is considered. This draws on a conservative approach, whereas no element of the original dataset is deleted and the corrections focus on adding data. This precondition makes sense in some contexts and is questionable in others. A detailed classification of contexts is outside the scope of this article. This limitation will be addressed in future work.

More formally, we can define the following.

Definition 4.1 (*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 $\Omega = \cup_{i_i \in I} \text{Reach}(i_i)$ be the set of all the states reachable by the FSS from the initial one;
—let $\pi = s_0 a_0, \ldots, s_i a_i, s_j$ be a reachable and *inconsistent trajectory*, that is, a trajectory where $s_0, s_i, \ldots, s_j \in \Omega$ and s_j is an inconsistent state (i.e., $s_j \in E$), while $s_0, \ldots, s_i \notin E$.

Then, a T*-cleansing action sequence* for the pair (s_i, a_i) is a nonempty sequence of actions $\epsilon^c = c_0, \ldots, c_n \in A$, such that exists a trajectory $\pi_c = s_0\ a_0, \ldots, s_{i-1}a_{i-1}, s_i c_0$, $s_{i+1}c_1$, …, $s_{i+n}c_n$, $s_{i+n+1}a_i$, $s_{i+n+2} \ldots$ on $\mathcal{S}$ with $|\epsilon^c| \leq T$ (the finite horizon), where all the states $s_0, \ldots, s_{i+n+2}$ are consistent.

Since the terms event and action are used interchangeably in this article as outlined in Section 3.2, by abuse of notation we will use the expression *cleansing event sequence* to refer to the events (or records) that can be viewed as the transcriptions of the actions $c_0, \ldots, c_n$.

In the remainder of the article, the term *correction* will also be used to refer to a cleansing event sequence (or cleansing action sequence).

In the AI planning field, a *universal plan* is a set of policies, computed off-line, able to bring the system to the goal from any feasible state (the reader can see Schoppers [1987], Cimatti et al. [1998], and Della Penna et al. [2012] for details). Similarly, we are interested in the synthesis of an object; we call UC, which summarizes for each *pair* (state, event) leading to an inconsistent state, the set A' of *all* the feasible cleansing event sequences. This UC is computed once and then can be used to cleanse any FSE-DB containing data of the domain for which the UC has been developed.

The UC is computed starting from a dataset and a consistency model. Considering the data, some domain attributes can be replaced by a symbolic version, similarly to what has been done for the Universal Checker. The replacement of domain values with symbolic data can provide a twofold effect: to limit the size of the UC and to make it general enough with respect to the data instances.

More formally, we define the Universal Cleansing Problem (UCP) and its solution as follows.

Definition 4.2 (*Universal Cleansing Problem and Universal Cleanser*). A UCP is a triple $\mathcal{D} = \{\mathcal{S}, E, T\}$, where $\mathcal{S} = (S, I, A, F)$ is an FSS, E is the set of error (or inconsistent) states identified by the model checker, and T is the finite horizon. Let Ω be the set of reachable states.

A solution for $\mathcal{D}$, or a UC for $\mathcal{D}$ is a *map* $\mathcal{K}$ from the set $\Omega \times A$ to a subset $\mathcal{A}'$ of the power set of A, namely, $\mathcal{A}' \subseteq 2^A$, where for each inconsistent trajectory $\pi = s_0a_0 \ldots s_ia_i$ s_j the set $\mathcal{A}'$ must contain *all the possible* T-cleansing event sequences (i.e., cleansing action sequences) for the pair (s_i, a_i) or $\mathcal{A}' = \emptyset$.

We recall that the terms event and action are used interchangeably in this article, as outlined in Section 3.2.

The pseudocode of the algorithm generating a UC is given in Algorithms 1 and 2. It has been implemented on the top of the UPMurphi tool [Della Penna et al. 2009]. Algorithm 1 takes as input a description of the consistency model and of the SDD (Symbolic Data Description, as outlined in Section 3.3), the Universal Checker (which can be viewed as a summary of the error codes and related knowledge), and a finite horizon T. Then, for each error code it looks for the feasible cleansing event sequences (according to Definition 4.1). This work is recursively accomplished by Algorithm 2, which explores the FSS (generated by a model checker from the consistency model and the SDD) through a depth-first visit collecting and returning the cleansing solutions, i.e., a description of the events required to turn the data sequence into a consistent one.

Consider again the cruise ship example of Table I. We use the SDD described in Table II, the consistency model described in Figure 1(a), and the Universal Checker shown in Table III. The cleansing event sequences are identified according to the following constraints:

—Loops are avoided on the cleansing event sequence. In the cruise ship domain a minimal correction has no loops on the path from the error state to the one where the last event is allowed. For example, two corrections of the error code 1 are shown (the underlined events are added to fix the error). The first one $pos = sea$, $\underline{(check\text{-}in, City_X)}$, $(check\text{-}out, City_X)$ is loop free and is the minimal correction of $pos = sea$, $\underline{(check\text{-}in, City_Y)}$, $\underline{(check\text{-}out, City_Y)}$, $\underline{(check\text{-}in, City_X)}$, $(check\text{-}out, City_X)$.

Table IV. The Universal Cleanser for the Cruise Ship Example

$\sim$ is for $\sim$-*ordered* (e.g., $d_1 \sim d_2 \equiv d_1 \leq d_2$). We recall that the UC is a map that assigns a set of corrections to each pair (reachable state, event). The error codes are not in the definition of UC, however, they have been added for the sake of clarity.

Error Code	([Reachable State],(Event))	Corrections (ShipID, City, Date, Event Type)
1	$([pos = sea], (\ldots, check\text{-}out, City_X))$	$(s, City_X, d, check\text{-}in)$
2	$([pos = harbor \wedge city = City_X], (\ldots, check\text{-}out, City_Y))$	$(s, City_X, d_1, check\text{-}out) \sim (s, City_Y, d_2, check\text{-}in)$
3	$([pos = harbor \wedge city = City_X], (\ldots, check\text{-}in, City_Y))$	$(s, City_X, d, check\text{-}out)$
4	$([pos = harbor \wedge city = City_X], (\ldots, check\text{-}in, City_X))$	$(s, City_X, d, check\text{-}out)$

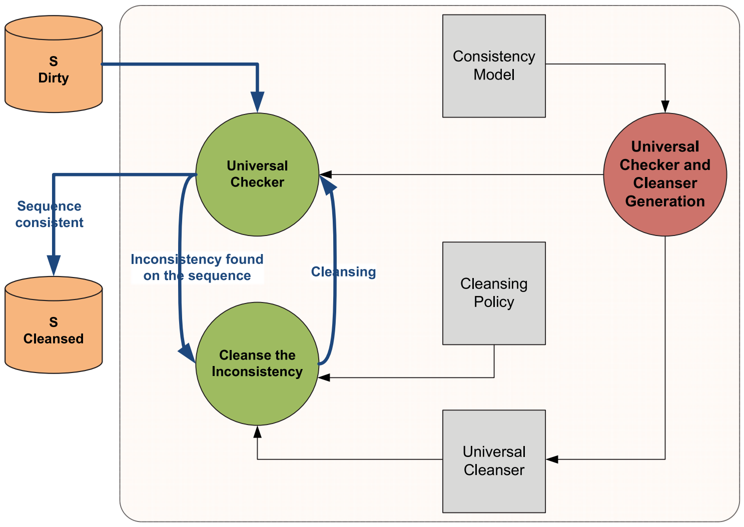

Fig. 2. A graphical representation of the consistency verification and cleansing process.

—Cleansing sequences are explored using a finite horizon $T = 2$, which is twice the diameter of the automaton of Figure 1(a). The automaton diameter is the number of hops of the longest path connecting two nodes (only loop free paths are considered).

Table IV shows the UC for our example; the entry list is minimal since symmetry reduction techniques have been used (e.g., the inconsistency cases ($[pos = sea]$, $(check\text{-}out, City_Y)$) and ($[pos = sea]$, $(check\text{-}out, City_X)$) can be similarly addressed), therefore the former is dropped. The UC, once generated, can be used to cleanse any kind of FSE-DB compliant with the model from which it has been generated. The example described in Table IV is very simple; every error code has only one possible cleansing sequence, therefore no selection task is required.

In more complex scenarios, several cleansing event sequences can be identified for a specific error code (i.e., an inconsistency can be cleansed in several ways). Therefore, a policy for selecting a cleansing sequence among the available ones is required. A more complex UC is described in Section 5, and a policy example is described in Section 4.2. An overview of the UC framework is shown in Figure 2.

Considering the case where several errors can be found in the same data partition, once the first CFP has been identified, it can be fixed in several ways (depending on the policy chosen), each one affecting the identification of further CFPs (e.g., a cleansing intervention can create a subsequent CFP, while a different intervention

ALGORITHM 1: UniversalCleansing

```
Input: $FSS\ \mathcal{S}$,
    Universal Checker UC
    Symbolic Domain Description SDD
    finite horizon $T$
Output: Universal Cleanser $\mathcal{K}$
1: $E \leftarrow \dots$ //List of error states derived from the Universal Checker UC
2: $level \leftarrow 0$; //to stop when $T$ is reached
3: for all $s^{err} \in E$ do
4:    for all $s \in S, a \in A$ such that $F(s, a) = s^{err}$ do
5:       $\mathcal{K}[s, a] \leftarrow$ auxUC$(s, a, level)$
6:    end for
7: end for
8: return $\mathcal{K}$
```

ALGORITHM 2: auxUC

```
Input: $s, a, level$
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 $a' \in A$ such that $F(s, a') = s'$ with $s' \notin E$ do
6:       if $F(s', a) = s''$ such that $s'' \notin E$ 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:          end for
15:       end if
16:    end for
17: end if
18: return $cs[\,]$
```

may generate no further CFP). The identification of reset events and the consequent (dataset) partition helps reducing interferences, since an intervention within a data partition will not propagate after the reset event. Unfortunately, reset events cannot completely avoid interferences since several CFPs can be found before a reset event. The policies selecting the cleansing intervention usually address this issue. A (complex) policy example is outlined in Section 4.2.

4.1. Consistency Verification and Cleansing Overview

Figure 2 describes the overall cleansing process. As a first step, a consistency model of the domain is defined, then the Universal Checker and Cleanser are automatically synthesized. The Universal Checker is used to verify 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 correction for the same inconsistency, a criterion (i.e., a *policy*) is required to select a suitable correction. Policies are domain dependent and they can change according to the analysis purposes (e.g., it is possible to select the shortest correction, the longest one, etc.). 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.”

Clearly, the cleansing results are policy dependent (i.e., the cleansed data are affected by the chosen policy). Although very interesting, the discussion on how to select a suitable policy for a specific task is not further explored in this work. Here, we outline that any kind of policy can be used once the UC has been generated, therefore several goals even antithetic can be pursued. Some policy examples will be presented in Section 4.2 and in Section 5.4. The interested reader can refer to Mezzanzanica et al. [2012] where the policy issues for maximizing and minimizing a reference indicator on the labor market domain are accurately described.

A UC, a policy, and an engine that can use them will be called *cleansing procedure* or *cleanser* hereafter.

4.2. A Policy Example: Maximizing and Minimizing an Indicator

This section will describe two cleansing policies developed for the cruise ship data introduced in Section 1. Let us suppose that the *active travel days* indicator is evaluated; this indicator computes the days spent on the sea. For example, considering the data of Table I, the ship $S01$ spent 14 days on the sea between the Venice check-out and the Lisbon check-in (the check-in and check-out dates were not counted because they were not entirely spent at sea).

The indicator computation is not a trivial task because of inconsistencies. The problem is addressed by the Universal Checker and the UC previously introduced: the former identifies the inconsistency, while the latter suggests one cleanse the data by adding a Lisbon check-out. Two cleansed versions of the original database are produced using two antithetic policies, which aim, respectively, to maximize and minimize the *active travel days* indicator. In this case, the policy scope is very limited, only the check-out date should be identified.

The two policies are exemplified as follows:

(1) The (Lisbon) missing departure date can range from 30th April 2011 to 5th May 2011; the bounds are given by the event dates surrounding the missing one. For simplicity, we are not taking into account the time a ship takes to physically travel from Lisbon to Barcelona, which would narrow the time slot.
(2) The (Lisbon) missing departure date is set on 30th April 2011 according to the policy maximizing the travel days; conversely, 5th May 2011 is chosen according to the policy minimizing the travel days.

All database inconsistencies are addressed in a similar way; then, the *active travel days* indicator mean value is computed on both the career datasets produced using the two policies. The difference between the two mean values can be used to estimate the impact of the inconsistencies.

The consistency model and the SDD of the labor market place described in Section 5 are quite more complex, therefore less trivial policies are required.

5. THE CASE OF “THE WORKER CAREERS ADMINISTRATIVE ARCHIVE”

This section will outline how the Universal Checker and Cleanser described in the previous sections were used in a real domain to cleanse the Italian labor data for analysis purposes. Since 1997, the Italian public administration has been collecting data concerning employment and active labor market policies [The Italian Ministry of Labour and Welfare 2012], generating an administrative archive useful for studying the labor market dynamics (e.g., see Martini and Mezzanzanica [2009]). The Italian labor law requires an employer to notify the public administration every time an employee is hired or an employment contract is modified (e.g., from part-time to full-time, or from fixed-term to unlimited-term). Such information is called *mandatory communications*.

Unfortunately, the data quality is very poor (e.g., see Cesarini et al. [2007]) and the framework described in the previous sections was used to assess and address inconsistency issues.

5.1. Domain Modeling

Each mandatory communication is stored in a record that can be broken down in 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. Event 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 or unlimited-term contract, etc.).
empr_id. Employer involved in the event.

The development of a consistent career path over time is described by a *sequence* of events ordered with respect to *e_date*, the sequence can be considered as longitudinal data. Considering the terminology introduced in Definitions 3.1 and 3.3, the ordered set of events for a given *w_id* is an FSE-D (a career), and the FSE-Ds union composes the FSE-DB (a set of careers). Now, let us closely look at career consistency, where the consistency semantic is derived from the Italian labor law, from domain knowledge, and from common practice. Some 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 (time) extension cannot change the existing contract type (*c_type*) or 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).

The FSS shown in Figure 3 is derived from the constraints just introduced. It outlines a consistent career development and it can be viewed as a consistency model. The FSS shows only the consistent transitions, while the inconsistent ones (transitions triggered by events that lead to an "error state"), the *conversion* and *extension* events, are not shown to improve readability. The model showed in Figure 3 also addresses some trivial constraints: an employee cannot have a *cessation* event for a company for which she/he does not work, an event cannot be recorded twice, and so on. These constraints are not further commented on.

The worker's career at a given time point (i.e., the system state) is described by three state variables: the list of companies for which the worker has an active contract ($C[]$), the working time (part-time, full-time) for each contract ($M[]$), and the list of contract types ($T[]$). To give an example, $C[0] = 12,\ M[0] = PT,\ T[0] = unlimited$ describes a worker having an active unlimited part-time contract with company 12.

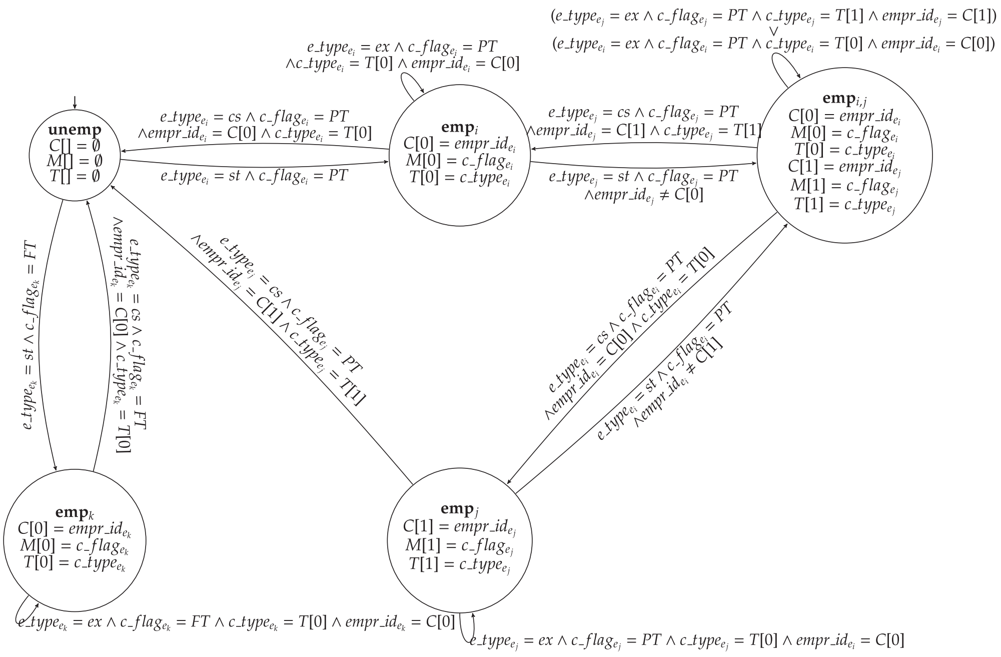
unemp
C[] = ∅
M[] = ∅
T[] = ∅
emp_i
C[0] = empr_id_{e_i}
M[0] = c_flag_{e_i}
T[0] = c_type_{e_i}
emp_{i,j}
C[0] = empr_id_{e_i}
M[0] = c_flag_{e_i}
T[0] = c_type_{e_i}
C[1] = empr_id_{e_j}
M[1] = c_flag_{e_j}
T[1] = c_type_{e_j}
emp_k
C[0] = empr_id_{e_k}
M[0] = c_flag_{e_k}
T[0] = c_type_{e_k}
emp_j
C[1] = empr_id_{e_j}
M[1] = c_flag_{e_j}
T[1] = c_type_{e_j}
e_type_{e_i} = ex ∧ c_flag_{e_i} = PT ∧c_type_{e_i} = T[0] ∧ empr_id_{e_i} = C[0]
(e_type_{e_j} = ex ∧ c_flag_{e_j} = PT ∧ c_type_{e_j} = T[1] ∧ empr_id_{e_j} = C[1]) ∨ (e_type_{e_i} = ex ∧ c_flag_{e_i} = PT ∧ c_type_{e_i} = T[0] ∧ empr_id_{e_i} = C[0])
e_type_{e_i} = cs ∧ c_flag_{e_i} = PT ∧empr_id_{e_i} = C[0] ∧ c_type_{e_i} = T[0]
e_type_{e_i} = st ∧ c_flag_{e_i} = PT
e_type_{e_j} = cs ∧ c_flag_{e_j} = PT ∧empr_id_{e_j} = C[1] ∧ c_type_{e_j} = T[1]
e_type_{e_j} = st ∧ c_flag_{e_j} = PT ∧empr_id_{e_j} ≠ C[0]
e_type_{e_k} = st ∧ c_flag_{e_k} = FT
e_type_{e_k} = cs ∧ c_flag_{e_k} = FT ∧empr_id_{e_k} = C[0] ∧ c_type_{e_k} = T[0]
e_type_{e_j} = cs ∧ c_flag_{e_j} = PT ∧empr_id_{e_j} = C[1] ∧ c_type_{e_j} = T[1]
e_type_{e_i} = cs ∧ c_flag_{e_i} = PT ∧empr_id_{e_i} = C[0] ∧ c_type_{e_i} = T[0]
e_type_{e_i} = st ∧ c_flag_{e_i} = PT ∧empr_id_{e_i} ≠ C[1]
e_type_{e_k} = ex ∧ c_flag_{e_k} = FT ∧ c_type_{e_k} = T[0] ∧ empr_id_{e_k} = C[0]
e_type_{e_j} = ex ∧ c_flag_{e_j} = PT ∧ c_type_{e_j} = T[0] ∧ empr_id_{e_i} = C[0]

Fig. 3. A graphical representation of a valid worker's career FSS where $st = start$, $cs = cessation$, $cn = conversion$, and $ex = extension$. The content of every node describes how the state evolves when an event happens (i.e., what are the values assigned to the state variables). The node *unemp* represents an unemployment state, the nodes emp_i and emp_j represent a single-part-time employment state, and the node $emp_{i,j}$ represents a two-part-time employment state.

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, $emp_{i,j}$).

5.2. From Actual to Symbolic Data

A mapping from actual to symbolic data has been identified, as described in Section 3.3, taking into account both the states and the events of the automaton of Figure 3.

—The attribute w_id is not used in the FSS described in Figure 3, and no total order relation is defined among its instances; therefore, a symbolic set composed by one element is chosen. Choosing a symbolic domain composed by one element reflects that the speculations about people careers can be performed by focusing on a single person.

—The e_id and the e_date attributes are not directly managed by the FSS. The automaton requires the events to be ordered from the time point of view, and this is already accomplished by the order relation enforced on the e_date attribute.

—The e_type, c_flag, c_type domain values are already bounded and have low cardinality, therefore they are used "as is."

—The empr_id attribute domain is mapped on a symbolic set of three symbols $\{empr_x, empr_y, empr_z\}$ according to the process described in Section 3.3. The automaton in Figure 3 has at most two state variables (i.e., $C[0]$ and $C[1]$ in the state emp_i_j) and the event e has one attribute referring to the employer domain; therefore, three symbols have been chosen.

Error Code	Reachable State	Event	Correction (Short Name)	Corrective Events (e_date, e_type, c_flag, c_type, empr_id)
289	$emp\_k[FT, Limited, Company_X]$	$(d_{cfp}, st, PT, Limited, Company_Y)$	Seq. 1	$(d, cs, FT, Limited, Company_X)$
			Seq. 2	$(d, cn, PT, Limited, Company_X)$
			Seq. 3	$(d, cn, PT, Unlimited, Company_X)$
			Seq. 4	$(d_1, cn, FT, Unlimited, Company_X) \sim (d_2, cs, FT, Unlimited, Company_X)$
			...	
			Seq. 9	$(d_1, cn, FT, Unlimited, Company_X) \sim (d_2, cn, FT, Limited, Company_X) \sim (d_3, cn, PT, Unlimited, Company_X)$

Fig. 4. Some corrective event sequences given by the UC for the error code 289. The attributes e_id and w_id are omitted for the sake of simplicity. d_1 and d_2 are date values. $\sim$ is for $\sim$-*ordered*.

Finally, we highlight that the model satisfies the conditions introduced in Section 3.3, namely, (1) despite a total order relation is defined for the empr_id domain, it is not considered in the automaton, and (2) there is no condition comparing a symbolic value and a nonsymbolic one.

5.3. A Cleanser for Careers Data

In this subsection, we describe how the Universal Checker and Cleanser (introduced in Sections 3 and 4) were used for investigating the worker careers data. An extract of the Universal Cleanser is shown in Figure 4.

—A Universal Checker and a UC were generated from the consistency rules described in Section 5.
—An FSE-D is partitioned into nonoverlapping subsets using the reset events introduced in Section 3.5. The *Full-time Start* and *Cessation* events can act as reset events according to the FSS depicted in Figure 3. The subsets are checked as if they were independent careers.
—Two different cleansing procedures have been built upon the UC using two cleansing policies: one selects the correction that maximizes the *working days* indicator, while the other one selects the correction that minimizes the same indicator. The two cleansers are called, respectively, $max_c()$ and $min_c()$. The *working days* indicator counts how many days a person has been employed over a time range. For the sake of simpleness holidays are not considered. More details on $max_c()$ and $min_c()$ are given in Section 5.4.

The upper bound value (UB) is the *working days* indicator computed on an FSE-D cleansed using the maximizing policy, while the lower bound value (LB) is the *working days* indicator achieved using the minimizing policy. The UB and LB identify the indicator possible value range.

For each FSE-D the UB (LB) value is computed, then the UB (LB) mean value is calculated over all the cleansed careers. The more inconsistencies are found in the database, the larger the gap between the UB and LB mean values will be. The gap can be interpreted as a measure of the uncertainty related to the data inconsistency. The algorithm computing the UB and LB values is described in the appendixes.

5.4. Labor Domain Cleansing Policies: An Overview

Once a UC has been created, a policy is required to build a *ready-to-go* cleanser and perform cleansing activities. A policy focuses on selecting the correction to perform (when several are available) and on identifying the field values of the records to be added (e.g., the event date). The complexity and the determinism of a cleanser (i.e., whether to cleanse data in a different order will get different final results) depends

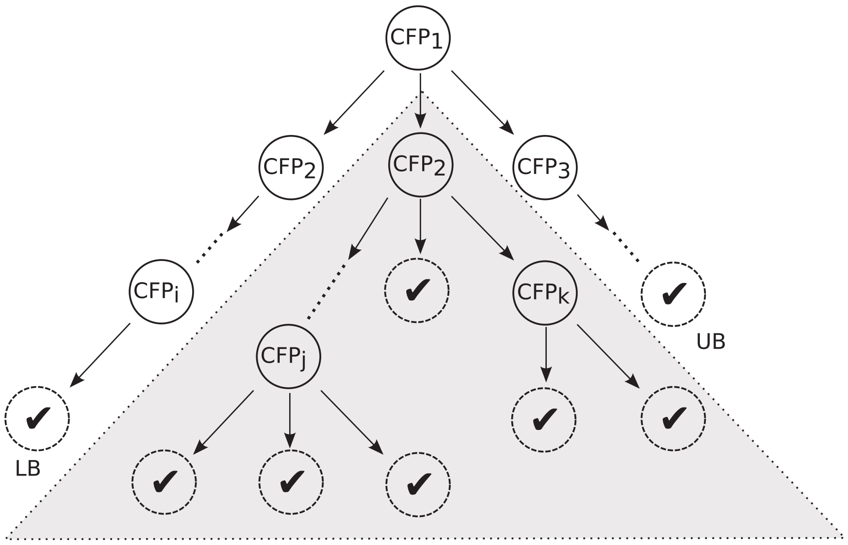

Fig. 5. An example of tree exploration for dataset correction. Leaf nodes represent a cleansed generated dataset, while non leaf nodes represent datasets where CFPs still need to be addressed.

on the chosen policy. These issues will be analyzed for the policy described in this subsection.

We recall that event cancellations are excluded from the correction managed in this article as outlined in Section 4.

The problem of identifying the indicator UB and LB values can be broken down into three steps: (1) identifying the CFP (if any), (2) selecting the possible corrections to perform from the UC, and (3) identifying the cleansing option that maximizes (minimizes) the reference indicator computed on the cleansed data.

—The set of CFPs and related cleansing interventions can be identified as follows: The event sequence is checked for inconsistency using the Universal Checker; when a CFP is found, a cleansed dataset version is derived by performing a cleansing intervention. Once an inconsistency has been cleansed, the consistency check is repeated again, since other CFPs can be detected in the remainder of the sequence. This process is represented in Figure 5. Whereas the root node is the original inconsistent dataset (the CFP found is described within the node), the inconsistency is fixed in its children (the CFP can be fixed in several ways and every child is a possible cleansed version of its father). Unfortunately, still another inconsistency can be found in the data represented by a child, which in turn is fixed in its respective children. A non leaf node represents an inconsistent dataset version (the CFP found is stated in the node), while leaf nodes are consistent versions of their ancestors.
—The cleansing selection can be described as the process of selecting the cleansing combinations so that in Figure 5 a path is selected from the root to a leaf node. The leaf node reached is the final cleansed version of the dataset.
—The "hunt" for UB and LB values terminates with the identification of two leaf nodes that, respectively, maximizes and minimizes the indicator.
—The tree shown in Figure 5 can grow exponentially and a complete exploration would be an intractable problem, considering the size of the data to be analyzed in a real case. Nevertheless, the research of the UB and LB cleansed datasets does not require exploring the whole tree, a heuristic approach can be used.

The policy for selecting the correction is implemented using the algorithm described in Mezzanzanica et al. [2012] (the simplified version is also known as "divide et impera"). More details are given in Appendix C. It is worth highlighting that the set of possible corrections has been identified using the UC in this work, while in Mezzanzanica et al. [2012] the corrections were identified by domain experts. Still, the domain experts validated the corrections identified by the UC and recognized that the latter was able to find more suitable corrections than they did in Mezzanzanica et al. [2012]. The simplified algorithm version requires two conditions to be met:

(1) The data sequence being evaluated must be partitioned into nonoverlapping subsequences that fully cover the original sequence.
(2) Data cleansing and the indicator computation must be parallelized over the subsequences identified in the previous step.

If both conditions hold, then the UB and LB indicator computation can be parallelized within the boundaries of each identified subsequence, thus greatly reducing the computational effort.

Now, the two conditions will be investigated. In the labor domain context, an FSE-D is partitioned using the reset points. The subsets obtained are further partitioned using the CFPs, i.e., a career is partitioned in subsets, each one ending either with a CFP, or with a reset point, or with the last career event. The working days indicator is an additive function with respect to the subsequences identified. The demonstration is trivial, therefore it has been omitted.

Now, the aim is to show that cleansing a subsequence has either no impact both on the previous (i.e., the antecedents) and on the following (i.e., the subsequent) subsequences, or every possible correction inside a subsequence has the same impact on the remaining future subsequences. These can be summarized by saying that a cleansing intervention should satisfy the properties 'preserving a common future" (with respect to other corrections) and "not altering the past," more formally described in Appendix C. As a side effect, satisfaction of the two properties ensures the determinism of the corrections (there is no determinism if applying corrections in a different order will produce different final results).

Unfortunately, not all the corrections identified by the UC satisfy the "not altering the past" and the "preserving a common future" properties (more details in Appendix C), and this can lead to computational problems. Therefore, when the simplified algorithm cannot be used on a career subsequence, an *on/off* estimation approach will be used instead of exploring all the feasible corrected instances. The time slot will be considered full of working days (to compute the UB value) or having 0 working days (to compute the LB values). Hereafter, the use of the "divide et impera" algorithm will be called the *computation approach*, while the use of the "on/off" estimation will be called the *estimation approach*.

The subsequences where the computation approach cannot be used are marked as *unsafe* and no more effort is spent in finding a correction. Also, the surrounding subsequences are marked as unsafe until the nearest reset points are met. In other words, the reset points restrain the uncertainty generated by the violation of the two properties. Before and after the surrounding reset points, the system state can be rebuilt with certainty, and this limits the impact that inconsistencies (and the cleansing activities) can have on surrounding subsequences.

The UB and LB values are computed for each FSE-D composing the FSE-DB using either the computation or the estimation approach. Then, the average values of all the FSE-D UB and LB values are computed and analyzed.

The complexity of the simplified search algorithm (the "dividit et impera" version) is briefly sketched.

Table V. Experimental Results on Careers Data

	Average Working Days			*Results*		
	Layer	$\overline{LB}$	$\overline{UB}$	Modified	Consistent	# of Careers
A	(a)	1,377	1,979	YES	YES	82,989
	(b)	1,383	1,383	NO	YES	130,500
	(c)	0	4,017	NO	NO	56
	(d)	0	4,017	YES	NO	887
B	(a)	1,633	1,942	YES	YES	61,301
	(b)	1,383	1,383	NO	YES	130,500
	(c)	0	4,017	NO	NO	56
	(d)	0	4,017	YES	NO	384

(A) Data computed on *all* careers (i.e., using both the *computation* and the *estimation* approaches) and (B) data computed only on *safe* careers (i.e., only the *computation* approach results are considered). Note that 4,017 are the days between 1st January 2000 and 31st December 2010 (11 years).

Let S be an FSE-D describing a person's career composed by $\epsilon = e_1, \ldots, e_n$ events. S is a sequence of events according to Definition 3.1. Let us suppose that S can be partitioned into m subsequences $\epsilon = \cup_{i=1}^{m} \epsilon_i$ as described in this subsection. Given k_i the complexity of exploring the tree related to ϵ_i, the complexity of the UB (and LB) identification is $Comp = \sum_{j=1}^{m} k_i$. It is worth recalling that the exploration tree of a subsequence is a (small) subset of the whole sequence tree (an example of the latter is shown in Figure 5). Given $k_{max} = \max(k_i)$ the maximum cost of exploring a subtree, the following holds: $Comp \leq mk_{max}$.

In the case where several small subsequences can be identified, each tree would be very small, therefore the k_{max} can be subsumed by a (small) constant. In the latter case, the complexity can be considered linear with respect to the number of identified subsequences.

5.5. Results

We tested both the Universal Checker and the UC on an administrative database having 214,432 careers (composed by 1,248,751 mandatory notifications) observed between 1st January 2000 and 31st December 2010 (11 years). The database collects data about the working careers of the inhabitants of an Italian area.

Table V reports the average value of the working days indicator layered by the *consistent* and *modified* flags. Part A considers all the careers (both *safe* and *unsafe*), and the results are computed using both the *computation* and the *estimation* approach. Part B of Table V excludes from the analysis all the *unsafe* careers (i.e., careers having at least one *unsafe subset*). The careers data are checked again using the universal checker after the cleansing. The *consistent* flag describes whether a career is consistent at the end of the correction process or not. The *modified* flag states that at least one correction has been performed on the career. Hence, looking at Table V, row (a) represents careers made consistent, while (b) are careers already consistent. Row (c) represents careers for which no correction was found, while the corrections were ineffective for careers in row (d). The latter are careers that would require events to be erased to fix the error. These errors cannot be addressed by the cleansers since it can only add events. It is worth noting that (b) refers to consistent careers that do not need a fix; consequently, there is no source of uncertainty and for this reason the UB and LB values are equal. We computed a Sensitivity Index SI of the working days variable as $SI = \frac{\sum((\overline{UB} - \overline{LB}) \cdot Num)}{\sum(\overline{LB} \cdot Num)}$. For the dataset that includes *unsafe careers,* we obtained a $SI = 18.2\%$. Differently, the SI for the dataset excluding *unsafe careers* is $SI = 7.3\%$. As expected, the estimation approach greatly contributes to enlarging the *working days* indicator upper and lower bound distance. The *SI* value could have been narrowed by exploiting a more fine-grained approach for estimating the indicator value on unsafe subsequences, but this topic is not further explored in this article.

We implemented Algorithms 3 and 4 in C++, using the BOOST library to interact with the Database Management System (DBMS). All the experiments were performed on a 32bit 2.2GHz CPU in about 1 hour using 200MB of RAM.

6. CONCLUSION AND FUTURE WORKS

This article describes how a model-based approach can be used to investigate data inconsistencies and to perform cleansing activities. A model describing the consistent evolution of longitudinal data is formalized for a specific domain, a symbolic description of data attributes is provided, then a model checker can generate a Universal Checker and a UC. The Universal Checker is a knowledge base that can be used to identify inconsistencies on the reference domain data. In a similar way, the UC is a knowledge base that can be used to identify the possible cleansing activities to address inconsistencies. The Universal Checker and UC can foster the development of data cleansing solutions since the development effort mostly focuses on designing the consistency model and on developing a policy for selecting the correction when several alternatives are available. In this way, the development of cleansing solutions can focus more on the model (what to do) rather than the implementation (how to achieve the desired goals).

We evaluated the approach proposed in the article on a real-world scenario, namely, the "Worker Careers Administrative Archive," which stores the labor data of millions of citizens living in an Italian area. We performed a sensitivity analysis using two different cleansers produced using the presented approach. The results confirm the usefulness of exploiting model-based verification and cleansing techniques in the data quality field. The approach described in this article is a powerful instrument in the hands of domain experts since it effectively supports the data cleansing processes improvement. The sensitivity analysis enabled by the proposed approach is a powerful instrument to evaluate how data inconsistencies may affect an indicator result computation. Furthermore, it contributes to the process of defining and formalizing domain aspects. As a side effect, a better comprehension of the data peculiarities is achieved.

As a future work we would like to remove the restrictions about event cancellation (i.e., we are investigating how to build a UC that can drop events). We would also like to explore temporal logic to express consistency rules. Furthermore, we are going to evaluate our cleansing solutions against soft-computing-based approaches (e.g., cleansing based on learning algorithms). For these reasons, an online dataset has been made available to the community (see Boselli et al. [2013] for further details). We are evaluating making available a tool implementing the approach proposed in this article to the data quality community. The NADEEF system [Dallachiesa et al. 2013] is a promising framework and we are evaluating building a plugin, so that the proposed approach can complement the existing approaches already implemented in NADEEF, providing the end user with a powerful instrument to tackle data quality challenges.

APPENDIXES

A. CONSTRAINTS-BASED FORMALISM COMPARISON

To the best of our knowledge, FDs, CFDs, and DCs have not been exploited for verifying and cleansing longitudinal data sequences. Indeed, catching inconsistencies over longitudinal data sequence (as in the example shown in Table A1) through FDs could be an awkward task as detailed in the following. First, we will focus on FDs; later we will deal with CFDs and DCs.

Table A1. A Career Example where P.T. is for Part Time. The semantic of the data is described in Section 5.

Event	Event Type	Employer-ID	Date
01	P.T. start	Firm 1	12/01/2010
02	P.T. start	Firm 2	31/03/2011
03	P.T. cessation	Firm 1	15/06/2011
04	P.T. start	Firm 3	1/10/2012
05	P.T. start	Firm 4	1/06/2013
...	...	...	...

Let $\mathcal{R} = (D_1, \ldots, D_l)$ be the schema of the database relation shown in Table A (the semantic and the domain have been described in Section 5), and let R be an instance of $\mathcal{R}$. Then, let $\sim$ be a *total order* relation over the records of R, such that exists a $\sim$-ordered sequence of records $\epsilon = e_1, \ldots, e_n$, where $R = \cup_{i=1}^{n} e_i$, and $\forall i, j \in \mathbb{N}, i < j, e_i \sim e_j$. Let *next*() be a function such that $next(e_i) = e_{i+1}$; let $P_k = R_1 \bowtie_{cond} R_2 \ldots \bowtie_{cond} R_k$ be the k^{th} self inner join of R with itself ($R = R_1 = R_2 = \cdots = R_k$; the subscripts are added to improve readability), where *cond* put subsequent tuples of R in a row, that is, $cond : next(e_{x,z}) = e_{x+1,z+1}$, where $e_{a,b}$ is the b^{th} record of the sequence of the relation R_a participating to the inner join. For simplicity, we assume that Table A1 reports data on only one career.[6]

According to our experience, the k^{th} self join allows one to express the constraints described in Section 5.1 on the data of Table A using functional dependencies; however, this approach has several drawbacks:

(1) An expensive join of k relations is required to check constraints on sequences of k elements.
(2) The higher the k value, the larger the attribute set to be considered for specifying the constraints.
(3) A modification of k requires one to rewrite the FD set.
(4) FDs do not provide any hint on how to fix inconsistencies, as discussed in Fan et al. [2010].

To give an example, the start and cessation of a part time (e.g., events 01 and 03) can happen arbitrarily many times in a career before the event 05 is met (where the inconsistency is detected), therefore there cannot be a k value high enough to surely catch the inconsistency described. Unfortunately, the arbitrarily (many times) repetition of events is not uncommon in longitudinal data. An upper bound value can be identified empirically (e.g., if the longest career has 30 events in the previous example, $k = 30$ is fine; however, a new dataset having a longer career will require one to change k and to rewrite the FD set).

The previously mentioned drawbacks still apply to CFDs and drawbacks 2, 3, and 4 hold also for DCs. DCs can be defined over an arbitrary set of tuples; most of the works focus on two tuples per time. In such a case, the kth self join would allow DCs to check at most $2k$ long sequences. In the context of CFDs the previous consideration about k can be rephrased as follows: there cannot be a $2k$ value high enough to surely catch the inconsistency described. The considerations just introduced can be easily generalized for DCs defined over more than two tuples.

These latter considerations show that FDs, CFDs, and DCs are not well suited for addressing the inconsistency problem outlined in Table A (and in Section 5), therefore

[6]The previously mentioned schema can be easily modified to manage the data of several careers in the same table and it can be also further enhanced to manage sequences shorter than k by using the left outer join instead of the inner join.

the existing tools and frameworks based on FDs, CFDs, and DCs are not useful in this context.

The model checking approach used in this article is limited by the finite horizon too (see Section 3.1), in a similar way to the previously mentioned k value, however, differently from FDs, CFDs, and DCs, the finite horizon is a parameter whose variation does not require changing the (consistency) model description.

B. UPPER AND LOWER BOUND IDENTIFICATION

Hereafter, the *ccheck* function will refer to a Universal Checker used to detect inconsistencies.

For the sake of clarity, the semantics of *ccheck* and *CFP* are formally defined as follows.

Definition B.1 (*ccheck*). Let S be an FSE-D and let $\epsilon = e_1, \ldots, e_n$ be a sequence of events according to Definition 3.1, then $ccheck$: FSE-D $\rightarrow \mathbb{N} \times \mathbb{N}$ returns the pair $\langle i, er \rangle$, where

(1) i is the index of a minimal subsequence $\epsilon_{1,i} = e_1, \ldots, e_i$ such that $\epsilon_{1,i+1}$ is inconsistent, while $\forall j : j \leq i \leq n$, the subsequences $\epsilon_{1,j}$ are consistent.
(2) er is zero if $\epsilon_{1,n}$ is consistent, otherwise it is a natural number that uniquely describes the inconsistency error code of the sequence $\epsilon_{1,i+1}$.

Then, we can define CFP_S as the index of the event *after* the shortest consistent subsequence. By abuse of notation, we denote the index i of the pair returned as $first\{ccheck(S)\}$, while $second\{ccheck(S)\}$ denotes the element er. According to this notation, $CFP_S = first\{ccheck(S)\} + 1$.

Definition B.2 (*CFP*). Let S be an FSE-D and let $\epsilon = e_1, \ldots, e_n$ be a sequence of events according to Definition 3.1. Moreover, let *ccheck* be a function as in Definition B.1, then $CFP_S = first\{ccheck(S)\} + 1$ is a CFP for the dataset S. Clearly, if $first\{ccheck(S)\} = n$, then S has no *CFP*.

An additive function is defined as follows:

THEOREM B.3 (ADDITIVE FUNCTION). *Let $\epsilon = e_1, \ldots, e_n$ be an FSE-D (according to Definition 3.2) that can be partitioned into event subsequences such that $\cup_{i=1}^{m} \epsilon_i = \epsilon$, and $\forall i, j$ with $i \neq j$, $\epsilon_i \cap \epsilon_j = \emptyset$, then a function F : FSE-D $\rightarrow \mathbb{Z}$ is additive if $F(\epsilon) = \sum_{i=1}^{m} F(\epsilon_i)$ with $m < n$.*

Now, it will be illustrated how the correction tree can be explored to identify the UB and LB cleansed datasets (an example of the correction tree is depicted in Figure 5); the algorithm pseudocode is described in Algorithms 3 and 4.

Algorithm 3 focuses on an FSE-DB S and, for each inconsistent FSE-D S_i, it calls Algorithm 4 to cleanse the dataset (line 8). Algorithm 4 takes as input the dataset S_i and its CFP_{S_i}. Then it generates the possible instances S'_i as a result of all the possible intervention activities. The consistency of each generated S'_i is checked as follows:

—S'_i is completely consistent (line 5). In this case it is added to $S_i^{consistent}$.
—S'_i is still inconsistent but a new emerged CFP occurs *later* than the previous one. That is, the previous inconsistency has been fixed, but a new one has emerged in the remainder of the sequence (line 6). Then, the algorithm recursively calls EXPLORECORRECTIONTREE on the new instance trying to fix it (line 7).
—S'_i is still inconsistent, but the correction has generated a new CFP before the previous one (i.e., the corrective action has made the situation worse). In this case, the corrective action is discarded (line 9).

At the end of this computation, the set $S_i^{consistent}$ contains all the consistent instances generated for S_i (i.e., the leaf nodes of Figure 5). Then, the LB and UB cleansed versions can be extracted.

ALGORITHM 3: MAIN ALGORITHM

```
Input: $S$ //An FSE-DB according to Definition 3.2
    $ccheck$ //According to Definition B.1
    $ind$ //An indicator satisfying the conditions of Theorem B.3
 1: for all $S_i \in S$ do
 2:    $lower_{S_i} \leftarrow \emptyset$; //global lower value for $S_i$
 3:    $upper_{S_i} \leftarrow \emptyset$; //global upper value for $S_i$
 4:    $S_i^{consistent} \leftarrow \emptyset$; //global set of the consistent versions of $S_i$
 5:    $CFP_{S_i} \leftarrow first\{ccheck(S_i)\} + 1$;
 6:    if ($CFP_{S_i} \leq n$) then
 7:       //$S_i$ is not consistent
 8:       EXPLORECORRECTIONTREE($S_i, CFP_{S_i}$); //Updates the $S_i^{consistent}$ set
 9:       $S_i^{UB} \leftarrow UBCorrectionIdentification(S_i^{consistent})$; //This selects the correction that
                                                   //maximizes the reference indicator
10:       $S_i^{LB} \leftarrow LBCorrectionIdentification(S_i^{consistent})$; //This selects the correction that
                                                   //minimizes the reference indicator
11:       $upper_{S_i} \leftarrow compute\_indicator(S_i^{UB})$;
12:       $lower_{S_i} \leftarrow compute\_indicator(S_i^{LB})$;
13:    else
14:       //No inconsistency is found
15:       $upper_{S_i} \leftarrow compute\_indicator(S_i)$;
16:       $lower_{S_i} \leftarrow upper_{S_i}$; //The indicator has only one possible value when
                                   //no error is found, therefore $lower_{S_i}$ and $upper_{S_i}$ are equals
17:    end if
18: end for
```

ALGORITHM 4: EXPLORECORRECTIONTREE

```
Input: $S_i, CFP_{S_i}$
 1: for all $a_i$ corrective action for $S_i$ do
 2:    $S_i' \leftarrow$ apply($a_i, S_i$);
 3:    $CFP_{S_i'} \leftarrow first\{ccheck(S_i')\} + 1$;
 4:    if ($CFP_{S_i'} = n + 1$) then
 5:       $S_i^{consistent} \leftarrow S_i^{consistent} \cup S_i'$;
 6:    else if ($CFP_{S_i'} > CFP_{S_i}$) then
 7:       EXPLORECORRECTIONTREE($S_i', CFP_{S_i'}$);
 8:    else
 9:       //Discard the corrective action
10:    end if
11: end for
```

C. (DIVIDE ET IMPERA) ALGORITHM VERSION

In order to define the next properties, we introduced a relation between subsequences ϵ_1, ϵ_2 such that $\epsilon_1 < \epsilon_2$ if and only if $\forall e_i \in \epsilon_1, e_j \in \epsilon_2, e_i < e_j$.

A formal definition of the condition *preserving a common future* is provided as follows:

CONDITION C.1 (PRESERVING A COMMON FUTURE). *Every correction of a CFP found inside a subsequence ϵ_i must share a common future in all the subsequences ϵ_k where $\epsilon_k > \epsilon_i$.*

Sharing a common future means that the set of CFPs found in the subsequences ϵ_k and the set of changes made by the respective corrective actions are completely shared by the possible corrections of ϵ_i.

A formal definition of the condition *not altering the past* is provided as follows:

CONDITION C.2 (NOT ALTERING THE PAST). *Every correction of a CFP in ϵ_i will not modify prior subsequences ϵ_p where $\epsilon_p < \epsilon_i$.*

LEMMA C.1 (PRUNING CONDITIONS). *If Theorem B.3 and Conditions C.1 and C.2 hold for an indicator to be evaluated on an FSE-D, then the UB/LB case search can exploit a "divide et impera" paradigm (i.e., it can be computed independently on each ϵ_i).*

The conditions just introduced ensure that the event sequence can be split into nonoverlapping subsets (which fully cover the original event sequence) where the the UB and LB values search can be performed locally (in isolation on each subset). The interested reader can refer to Mezzanzanica et al. [2012] for more details.

In the labor domain described in Section 5 the Conditions C.1 and C.2 are met if two sufficient conditions hold:

CONDITION C.3 (NOT ALTERING THE PAST—SUFFICIENT). *A correction for a subsequence ending with a CFP does not affect previous subsequences (this ensures that condition C.1 is met).*

CONDITION C.4 (PRESERVING A COMMON FUTURE—SUFFICIENT). *The corrections done into a subsequence will lead to the same final state.*

Condition C.4 is enough to "preserve a common future" since the FSS modeling the consistent data behavior is a memoryless system [Csiszár and Körner 1981; Iosifescu 1980]).

The condition C.3 can be tested empirically during the dataset cleansing, while condition C.4 can be checked in advance for each entry of the UC.

Unfortunately, some corrections of the UC for the worker data do not satisfy the two sufficient conditions. An example is shown, as follows. Let us suppose that a person is in the unemployed status and then three part-time contracts are activated in line with different companies (e.g., on 23rd September (first event), on 15th October (second event), and on 18th November (third event), activated, respectively, with $company_x$, $company_y$, and $company_z$). Since a worker cannot have more than two part-time contracts at the same time, according to the business rules described in Section 5.1, either the part-time with $company_x$ or the one with $company_y$ was closed and the event was not recorded. Possible corrections are to close either the first or the second contract, or both. The "common future" is not preserved because the three possible corrections can bring the FSS of Figure 3, respectively, to the state emp_i, emp_j, or $unemp$. Generally speaking, choosing to close either the part-time with $company_x$ or the one with $company_y$, or both, may create different versions of the future. A new CFP will emerge if the contract with $company_x$ is closed (as a correction) and later a real cessation with $company_x$ is found. On the contrary, adding a cessation with $company_y$ will raise no problem in the previously described case. It is worth highlighting that the real cessation with $company_x$ may be unknown at the time when the correction is performed.

REFERENCES

James Abello, Panos M. Pardalos, and Mauricio G. C. Resende. 2002. *Handbook of Massive Data Sets*. Kluwer Academic, Norwell, MA.

Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking (Representation and Mind Series)*. MIT Press, Cambridge, MA.

Francesco Bartolucci, Alessio Farcomeni, and Fulvia Pennoni. 2013. *Latent Markov Models for Longitudinal Data*. Chapman & Hall/CRC, Boca Raton, FL.

Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. 2009. Methodologies for data quality assessment and improvement. *ACM Comput. Surv.* 41, 3 (July 2009), Article 16, 52 pages.

Carlo Batini and Monica Scannapieco. 2006. *Data Quality: Concepts, Methodologies and Techniques*. Springer, Berlin.

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

Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. 2003. Bounded model checking. *Adv. Comput.* 58 (2003), 117–148.

Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional functional dependencies for data cleaning. In *Proceedings of the IEEE 23rd International Conference on Data Engineering (ICDE 2007)*. IEEE, Turkey, 746–755. DOI:http://dx.doi.org/10.1109/ICDE.2007.367920

Roberto Boselli, Mirko Cesarini, Fabio Mercorio, and Mario Mezzanzanica. 2013. Inconsistency knowledge discovery for longitudinal data management: A model-based approach. In *Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data*, Andreas Holzinger and Gabriella Pasi (Eds.). Lecture Notes in Computer Science, Vol. 7947. Springer, Berlin, 183–194. DOI:http://dx.doi.org/10.1007/978-3-642-39146-0_17

Roberto Boselli, Mirko Cesarini, Fabio Mercorio, and Mario Mezzanzanica. 2014a. Planning meets data cleansing. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, Steve Chien, Minh Do, Alan Fern, and Wheeler Ruml (Eds.). AAAI Press, Portsmouth, NH,, 439–443. http://hdl.handle.net/10281/52130

Roberto Boselli, Mirko Cesarini, Fabio Mercorio, and Mario Mezzanzanica. 2014b. A policy-based cleansing and integration framework for labour and healthcare data. In *Interactive Knowledge Discovery and Data Mining in Biomedical Informatics*, Andreas Holzinger and Igor Jurisica (Eds.). Lecture Notes in Computer Science, Vol. 8401. Springer, Berlin, , 141–168. DOI:http://dx.doi.org/10.1007/978-3-662-43968-5_8

Roberto Boselli, Mirko Cesarini, Fabio Mercorio, and Mario Mezzanzanica. 2014c. Towards data cleansing via planning. *Intelligenza Artificiale* 8, 1 (1 2014), 57–69. DOI:http://dx.doi.org/10.3233/IA-140061

Mirko Cesarini, Mario Mezzanzanica, and Mariagrazia Fugini. 2007. Analysis-sensitive conversion of administrative data into statistical information systems. *J. Cases Inf. Technol.* 9, 4 (2007), 57–81.

Eun-Hye Choi, Tatsuhiro Tsuchiya, and Tohru Kikuno. 2006. Model checking active database rules under various rule processing strategies. *IPSJ Digital Courier* 2, 0 (2006), 826–839.

Jan Chomicki. 1995. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst. (TODS)* 20, 2 (1995), 149–186.

Jan Chomicki and Jerzy Marcinkowski. 2004. On the computational complexity of minimal-change integrity maintenance in relational databases. In *Inconsistency Tolerance*, Leopoldo Bertossi, Anthony Hunter, and Torsten Schaub (Eds.). Springer-Verlag, Berlin, 119–150. http://dl.acm.org/citation.cfm?id=2137582.2137587

Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *Proc. VLDB Endow.* 6, 13 (Aug. 2013), 1498–1509.

Alessandro Cimatti, Marco Roveri, and Paolo Traverso. 1998. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98) and the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI'98).* AAAI Press, Madison, WI, 875–881.

Edmund M. Clarke, Daniel Kroening, and Karen Yorav. 2003. Behavioral consistency of c and verilog programs using bounded model checking. In *Proceedings of the 40th Annual Design Automation Conference (DAC'03)*. ACM, New York, NY, 368–371. DOI:http://dx.doi.org/10.1145/775832.775928

Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. 1999. *Model Checking*. MIT Press, Cambridge, MA.

CMurphi Web Page. 2011. Homepage. Retrieved from http://www.dsi.uniroma1.it/tronci/cached.murphi.html.

Imre Csiszár and János Körner. 1981. *Information Theory: Coding Theorems for Discrete Memoryless Systems*. Vol. 244. Academic Press, New York.

Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: A commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data(SIGMOD'13)*. ACM, New York, NY, 541–552. DOI:http://dx.doi.org/10.1145/2463676.2465327

Giuseppe Della Penna, Benedetto Intrigila, Daniele Magazzeni, and Fabio Mercorio. 2009. UPMurphi: A tool for universal planning on PDDL+ Problems. In *Proceedings of ICAPS 2009*. AAAI Press, 106–113.

Giuseppe Della Penna, Benedetto Intrigila, Daniele Magazzeni, Fabio Mercorio, and Enrico Tronci. 2011. Cost-optimal strong planning in non-deterministic domains. In *Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics (ICINCO'11)*. SciTePress, Noordwijkerhout, The Netherlands, 56–66. DOI:http://dx.doi.org/10.5220/0003448200560066

Giuseppe Della Penna, Daniele Magazzeni, and Fabio Mercorio. 2012. A universal planning system for hybrid domains. *Appl. Intell.* 36, 4 (2012), 932–959. DOI:http://dx.doi.org/10.1007/s10489-011-0306-z

Giuseppe Della Penna, Daniele Magazzeni, Alberto Tofani, Benedetto Intrigila, Igor Melatti, and Enrico Tronci. 2008. Automated generation of optimal controllers through model checking techniques. In *Informatics in Control Automation and Robotics*, JuanAndrade Cetto, Jean-Louis Ferrier, Jos Miguel Costa dias Pereira, and Joaquim Filipe (Eds.). Lecture Notes in Electrical Engineering, Vol. 15. Springer, Berlin, 107–119. DOI:http://dx.doi.org/10.1007/978-3-540-79142-3_10

Sarv Devaraj and Rajiv Kohli. 2000. Information technology payoff in the health-care industry: A longitudinal study. *J. Manage. Inf. Syst.* 16, 4 (2000), 41–68.

Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.* 19, 1 (2007), 1–16.

Wenfei Fan. 2008. Dependencies revisited for improving data quality. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, New York, NY, 159–170. DOI:http://dx.doi.org/10.1145/1376916.1376940

Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2010. Towards certain fixes with editing rules and master data. *Proceedings of the VLDB Endowment* 3, 1–2 (2010), 173–184.

Craig Fisher, Eite Lauría, Shobha Chengalur-Smith, and Richard Wang. 2012. *Introduction to Information Quality*. Authorhouse, United Kingdom.

Craig W. Fisher and Bruce R. Kingma. 2001. Criticality of data quality as exemplified in two disasters. *Inf. Manage.* 39, 2 (Dec. 2001), 109–116.

Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2013. The llunatic data-cleaning framework. *Proc. VLDB Endow.* 6, 9 (2013), 625–636.

Fausto Giunchiglia and Paolo Traverso. 2000. Planning as model checking. In *Recent Advances in AI Planning*, Susanne Biundo and Maria Fox (Eds.). Lecture Notes in Computer Science, Vol. 1809. Springer, Berlin, 1–20. DOI:http://dx.doi.org/10.1007/10720246_1

Preben Hansen and Kalervo Järvelin. 2005. Collaborative information retrieval in an information-intensive domain. *Inf. Process. Manage.* 41, 5 (2005), 1101–1119.

Donald Hedeker and Robert D. Gibbons. 2006. *Longitudinal Data Analysis*. Vol. 451. Wiley-Interscience, Hoboken, NJ.

Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. 1997. Hytech: A model checker for hybrid systems. *Int. J. Softw. Tools Technol. Transf. (STTT)* 1, 1 (1997), 110–122.

Andreas Holzinger. 2012. On knowledge discovery and interactive intelligent visualization of biomedical dataChallenges in human-computer interaction and biomedical informatics. In *DATA*, Markus Helfert, Chiara Francalanci, and Joaquim Filipe (Eds.). SciTePress, Rome, 5–16.

Andreas Holzinger and Mario Zupan. 2013. KNODWAT: A scientific framework application for testing knowledge discovery methods for the biomedical domain. *BMC Bioinf.* 14 (2013), 191.

Marius Iosifescu. 1980. *Finite Markov Processes and their Applications*. Wiley, New York.

Solmaz Kolahi and Laks V. S. Lakshmanan. 2009. On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory (ICDT'09)*. ACM, New York, NY, 53–62. DOI:http://dx.doi.org/10.1145/1514894.1514901

David Lee and Mihalis Yannakakis. 1996. Principles and methods of testing finite state machines—A survey. *Proc. IEEE* 84, 8 (Aug. 1996), 1090–1123. DOI:http://dx.doi.org/10.1109/5.533956

Pietro Giorgio Lovaglio and Mario Mezzanzanica. 2013. Classification of longitudinal career paths. *Qual. Quant.* 47, 2 (2013), 989–1008.

Daniele Magazzeni. 2011. A framework for the automatic synthesis of hybrid fuzzy/numerical controllers. *Appl. Soft Comput.* 11, 1 (2011), 276–284.

Mattia Martini and Mario Mezzanzanica. 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*, C. Larsen, M. Mevius, J. Kipper, and A. Schmid (Eds.). Rainer Hampp Verlag, Mering, Germany.

Mario Mezzanzanica, Roberto Boselli, Mirko Cesarini, and Fabio Mercorio. 2012. Data quality sensitivity analysis on aggregate indicators. In *Proceedings of the International Conference on Data Technologies and Applications (DATA'12)*, M. Helfert, C. Francalanci, and J. Filipe (Eds.). SciTePress, Rome, Italy, 97–108. DOI:http://dx.doi.org/10.5220/0004040300970108

C. Norris Ip and D. L. Dill. 1996. Better verification through symmetry. *Formal Meth. Syst. Des.* 9, 1 (1996), 41–75.

Paolo Papotti, Xu Chu, and Ihab F. Ilyas. 2013. Holistic data cleaning: Putting violations into context. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE'13)*. IEEE Computer Society, Washington, DC, 458–469. DOI:http://dx.doi.org/10.1109/ICDE.2013.6544847

Anita Prinzie and Dirk Van den Poel. 2011. Modeling complex longitudinal consumer behavior with dynamic bayesian networks: An acquisition pattern analysis application. *J. Intell. Inf. Syst.* 36, 3 (2011), 283–304.

Erhard Rahm and Hong Hai Do. 2000. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.* 23, 4 (2000), 3–13.

Indrakshi Ray and Indrajit Ray. 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)*. Springer-Verlag, London, 266–279. http://portal.acm.org/citation.cfm?id=646045.676458

Thomas C. Redman. 1998. The impact of poor data quality on the typical enterprise. *Commun. ACM* 41, 2 (Feb. 1998), 79–82. DOI:http://dx.doi.org/10.1145/269012.269025

Marcel Joachim Schoppers. 1987. Universal plans of reactive robots in unpredictable environments. In *Proceedings of IJCAI 1987.* Morgan Kaufmann, 1039–1046.

Judith D. Singer and John B. Willett. 2003. *Applied Longitudinal Data Analysis: Modeling Change and Event Occurrence*. Oxford University Press, Cary, NC.

Diane M. Strong, Yang W. Lee, and Richard Y. Wang. 1997. Data quality in context. *Commun. ACM* 40, 5 (May 1997), 103–110. DOI:http://dx.doi.org/10.1145/253769.253804

Sing What Tee, Paul L. Bowen, Peta Doyle, and Fiona H. Rohde. 2007. Data quality initiatives: Striving for continuous improvements. *Int. J. Inf. Qual.* 1, 4 (2007), 347–367.

The Italian Ministry of Labour and Welfare. 2012. Annual Report about the CO System. Retrieved from http://www.cliclavoro.gov.it/Barometro-Del-Lavoro/Documents/Rapporto_CO/Executive_summary.pdf. (2012).

Moshe Y. Vardi. 1985. Fundamentals of dependency theory. IBM Research Report RJ4858.

B. L. William Wong, Kai Xu, and Andreas Holzinger. 2011. Interactive visualization for information analysis in medical diagnosis. In *Information Quality in e-Health*, Andreas Holzinger and Klaus-Martin Simonic (Eds.). Lecture Notes in Computer Science, Vol. 7058. Springer, Berlin, Heidelberg, 109–120. DOI:http://dx.doi.org/10.1007/978-3-642-25364-5_11

Mohamed Yakout, Mikhail J. Atallah, and Ahmed Elmagarmid. 2012. Efficient and practical approach for private record linkage. *J. Data Inf. Qual.* 3, 3 (Aug. 2012), Article 5, 28 pages. DOI:http://dx.doi.org/10.1145/2287714.2287715

Mohamed Yakout, Laure Berti-Équille, and Ahmed K. Elmagarmid. 2013. Don't be scared: Use scalable automatic repairing with maximal likelihood and bounded changes. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*. ACM, New York, NY, 553–564. DOI:http://dx.doi.org/10.1145/2463676.2463706

Received September 2013; revised April 2014; accepted July 2014