TOWARDS THE USE OF MODEL CHECKING FOR PERFORMING DATA CONSISTENCY EVALUATION AND CLEANSING

(Completed-paper)

Mario Mezzanzanica
Department of Statistics and Quantitative Methods – CRISP Research Centre – University of Milano-Bicocca
Mario.Mezzanzanica@unimib.it

Mirko Cesarini
Department of Statistics and Quantitative Methods – CRISP Research Centre – University of Milano-Bicocca
Mirko.Cesarini@unimib.it

Fabio Mercorio
CRISP Research Centre – University of Milano-Bicocca
Fabio.Mercorio@unimib.it

Roberto Boselli
Department of Statistics and Quantitative Methods – CRISP Research Centre – University of Milano-Bicocca
Roberto.Boselli@unimib.it

Abstract: This paper explores the application of formal methods (specifically, model checking) to the field of data quality. A model expressing the consistency of longitudinal data is derived from the domain knowledge. This model is used (1) to automatically verify the consistency of the data stored on a database and (2) to automatically generate a universal cleanser, i.e. a cleanser which summarises all the feasible corrections for any kind of inconsistency which may affect the data (as far as they can be guessed from the formal consistency model). The universal cleanser represents a repository of corrective interventions useful to develop cleansing routines. We applied our approach to a real world scenario: a formal verification has been performed on labour market data evaluating the consistency of people working careers. The results show that the proposed approach can improve the data quality evaluation and the development of cleansing activities.

Key Words: Data Consistency, Data Cleansing, Model Checking.

1 INTRODUCTION AND CONTRIBUTION

The ongoing relations between citizens and public administrations generate a lot of data and the administrative archives store a relevant portion thereof. Such data can be very valuable for supporting the decision making processes in several contexts: design, implementation, and evaluation of active policies, service design and improvement, etc. Some archives record also data along time, therefore they can be considered a source of longitudinal data (also called panel data), i.e. a set of (repeated) observations of

the same subjects along the time. For more details on longitudinal data see [14]. Several studies report that the *data quality* of enterprise and public administration databases is very low, e.g. [12, 2]. The organisations are getting more and more aware of the consequences and costs, therefore several plans, strategies, and actions have been implemented, e.g. as described in [15]. *Data quality* is a broad concept (a complete survey can be found in [2]). Here we focus on the *consistency* dimension which refers to the violation of semantic rules defined over a set of data items.

In this paper, a data consistency model is built from the domain knowledge, then a model checker can be used for verifying the consistency of longitudinal data and for generating the possible cleansing actions. An example is provided: the dataset in Tab. 1 shows a cruise ship travel plan. The ship usually travels by sea and stops at the port of calls (intermediate destinations), making a *checkin* when entering a harbour and a *checkout* when exiting. The reader will notice that the departure date from Lisbon is missing, since a *checkout* is necessary before entering the subsequent harbour (Barcelona). In this respect, the dataset is inconsistent.

EventId	ShipID	City	Date	Event Type
e_1	S01	Venice	12th April 2011	checkin
e_2	S01	Venice	15st April 2011	checkout
e_3	S01	Lisbon	30th April 2011	checkin
e_4	S01	Barcelona	5th May 2011	checkin
e_5	S01	Barcelona	8nd May 2011	checkout
...	...	...	...	...

Table 1: Travel Plan of a Cruise Ship

Data cleansing can be performed in several ways, nevertheless when no different (and more trusted) data source is available, the only feasible solution is to exploit business rules, i.e. to implement cleansing algorithms fixing inconsistencies using domain derived knowledge. The uncertainty affecting the data can impact on the aggregate data and on the information derived for decision making purposes, therefore the inconsistencies should be appropriately managed.
The comparison among archive contents and real data is often an unfeasible or very expensive option (e.g. due to the lack of alternative data sources, the cost of collecting the real data, etc.). On the contrary data assessment and cleansing based on business rules is frequently an effective and valuable solution.

In this paper we show how longitudinal data consistency can be modelled and verified through explicit model checking techniques. Once a model has been defined, a model checker can be used for deriving the set of possible errors and the set of possible corrective actions. These can be exploited: (1) for verifying the data consistency of real world archives and (2) as a foundation to partially automate the development of cleansing routines. It is worth to note that the approach presented in this paper bounds the effort of consistency checking to the formalisation of a suitable consistency model. Then, the task of performing the consistency check and the cleansing activities can be automatically executed.
We successfully applied model-checking-based techinques to assess the quality of an administrative archive.

The paper is organised as follows: in Sec. 2 the related works are surveyed; in Sec. 3 we shortly introduce model checking on finite state systems and how model checking can be used for verifying data consistency; Sec. 4 introduces the concept of the universal cleanser and provides an algorithm to compute it; in Sec. 5 we show some experimental results obtained working on a big administrative archive managing labour market information; finally, in Sec. 6 we report the conclusions and the future work.

2 RELATED WORK

Data quality has been addressed in different research domains including statistics, management, and computer science as reported in [27, 4]. For the sake of clarity, the works surveyed in this section have been classified into three groups according to the (main) goal pursued: *record linkage*, *error localisation and correction*, and *consistent query answering*. The classification adopted is not strict since several works could be classified in several groups.

Record linkage (known as *object identification*, *record matching*, *merge-purge problem*) aims to bring together corresponding records from two or more data sources or finding duplicates within the same one. The record linkage problem falls outside the scope of this paper, therefore it is not further investigated.

Error localisation and correction works can be further classified in: 1) those exploiting machine learning methods and 2) those exploiting data dependencies (formalised by domain experts) to detect and correct errors. Considering the latter, the effort of domain experts is required to formalise rules.

1) *Machine learning methods* can be used for error localisation and correction. Possible techniques and approaches are: unsupervised learning, statistical methods, data profiling, range and threshold checking, pattern recognition, clustering methodologies [23]. It is well known that these methods can improve their performance in response to human feedbacks, however the model resulting from the training phase can't be easily accessed and interpreted by domain experts. In this paper we explore a different approach where the consistency models are explicitly built and validated by domain experts.

2) *Dependencies based methods*. Several approaches focus on integrity constraints for identifying errors, however they cannot address complex errors or several inconsistencies commonly found in real data [18, 21].
Other constraint types have been identified in the literature: multivalued dependencies, embedded multivalued dependencies, and conditional functional dependencies. Nevertheless, according to Vardi in [33] there are still semantic constraints that cannot be described.
In [3] a context-free-grammar based framework is used to specify production rules (e.g., Univ. → University), to reconcile the different representations of the same concept. Such approach mainly focuses on the attribute level, whilst the work presented in this paper focuses on set-of-records consistency.
Works on *database repair* focus on finding a consistent and *minimally different* database from the original one, however the authors of [11] state that computational issues affect the algorithms used for performing minimal-change integrity maintenance.
Deductive databases [25] add logic programming features to relational systems and can be used for managing consistency constraints. To the best of our knowledge, few works in the literature focus on deductive databases and data quality: [29, 19]. Furthermore, scalability issues have to be investigated when dealing with large sets of data.
In [10] database triggers are derived from dynamic constrains expressed in a time (first-order) logic variant. However triggers can raise computational issues when processing large datasets.

Consistent query answering works, e.g. [6], focus on techniques for finding out *consistent answers* from inconsistent data, i.e. the focus is on automatic query modifications and not on fixing the source data. An answer is considered consistent when it appears in every possible repair of the original

database. Semantic constraints are expressed using functional dependencies. Basically already with two Functional Dependencies the problem of computing Consistent Query Answers involving aggregate queries becomes NP-complete [6].

Other works and tools not included in the previous categories are now briefly surveyed. The application of automata theory for inference purposes was deeply investigated in [34] in the database domain. The problem of checking (and repairing) several integrity constraint types has been analyzed in [1]. Unfortunately most of the approaches adopted can lead to hard computational problems. Formal verification techniques were applied to databases, to formally prove the termination of triggers [9], for semistructured data retrieval [24], and to solve queries on semistructured data [17].
Many data cleansing toolkits have been proposed for implementing, filtering, and transforming rules over data. A detailed survey of those tools is outside the scope of the paper. The interested reader can refer to [21].

3 FROM DATA CONSISTENCY VERIFICATION TO MODEL CHECKING

Model checking [6] is a hardware/software verification technique to verify the correctness of a suitably modelled system. The model is described in terms of *state variables*, whose evaluation determines a state, and *transition relations* between states, which specify how the system can move from a state to the next one as a consequence of a given input action. Focusing on *explicit* model checking techniques, a model checker verifies if a state transition system (i.e., the model) satisfies a property by performing an exhaustive search in the system state-space (i.e., the set of all the possible system states). The model checker exploits techniques to reduce or compress the system state-space to be analysed, e.g. the reachability analysis: the state variable values that can be actually reached are identified, the reachable ones are analysed while the others are not (although being in the range of the admissible values).
The system model to be verified is expressed by means of a model checking language. Then the model checker generates a corresponding Finite State System (FSS) where the desired consistency properties can be evaluated. For the sake of completeness, we highlight that model checking languages can describe both an FSS and an *implicit representation* (i.e. abstract and general) of some FSSs. An implicit representation can be translated into an FSS, and the verification is always performed on the latter. Due to the space limitations, we do not formalise such implicit representation of FSSs. However, the reader can see [4] where such concept is expressed by means of *Extended* Finite State Machines.

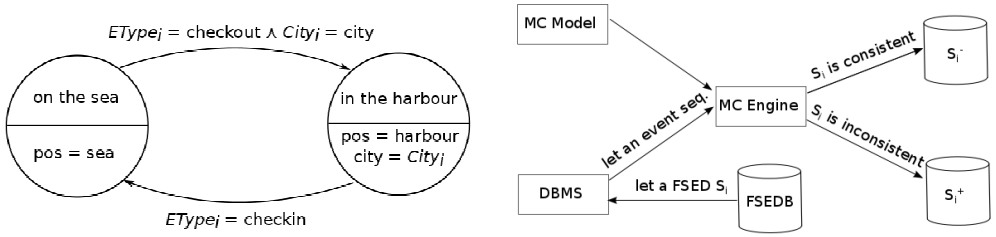
Definition 3.1 (Finite State System) *A Finite State System (FSS S) is a 4-tuple (S,I,A,F), where: S is a finite set of states, $I \subseteq S$ is a finite set of initial states, A is a finite set of actions and $F: S \times A \rightarrow S$ is the transition function, i.e. $F(s,a) = s'$ iff the system from state s can reach state s' via action a.*
Hence, a trajectory is a sequence of *state, action* $\pi = s_0 a_0 s_1 a_1 s_2 a_2 \ldots s_{n-1} a_{n-1} s_n$ where, $\forall i \in [0, n-1]$, $s_i \in S$ is a state, $a_i \in A$ is an action and $F(s_i,a_i)=s_{i+1}$.

Let S be an FSS according to Def. 3.1 and let φ be an invariant condition specifying some properties to be satisfied (called *safety properties* in the model checking domain) a state $s_E \in E$ is an error state if the invariant formula φ is not satisfied. Then, we can define the set of *error states* $E \subseteq S$ as the union of the states violating φ. We limit the error exploration to at most *T* actions (the finite horizon), i.e. only sequences reaching an error $s_E \in E$ within the finite horizon are detected. Note that this restriction has a limited practical impact in our contexts although being theoretically quite relevant.
Model checking is traditionally used to explore and verify all the feasible execution paths of a system. Then, informally speaking a *model checking problem* is composed by a description of the FSS to be explored, an invariant to verify, and a finite horizon. A *feasible solution* (if any) is a trajectory leading the system from an initial state to an error one.

3.1 Finite State Events Database

In an event-driven architecture, an *exogenous event* represents a change that may occur in the system configuration due to an external occurrence. A connection can be established between event-driven systems and databases containing longitudinal data: a database record (or a subset thereof) can be seen as an *event* arriving from the external world, and an ordered set of records can be seen as an *event sequence* (or action sequence). More precisely:

Definition 3.2 (Event, Event Sequence, and Finite State Event Dataset) *Let $R=(R_1,\ldots,R_n)$ be a schema relation of a database, let $e=(r_1, \ldots, r_m)$ be an event where $r_1 \in R_1, \cdots, r_n \in R_n$, then e is a record of the projection $(R_1, \ldots, R_m)$ over R with $m \leq n$.*
A *total order* relation ~ on events can be defined such that $e_1 \sim e_2 \sim \ldots \sim e_n$. An *event sequence* (or *action sequence*) is a ~-ordered sequence of events $\varepsilon=e_1, \ldots, e_n$. A *Finite State Event Dataset* is a *longitudinal* dataset extracted from a database that can be expressed as an event sequence.



(a) (b)

Figure 1: (a) A Graphical representation of the Cruise Ship Travel Plan model. The lower part of a node describes how the system state evolves when an event happens. (b) A Graphical representation of a process where the model checker is used to verify an FSEDB consistency.

Intuitively, the application of model checking to data quality problems is driven by the idea that a *model* describing the consistent evolution of *feasible* event sequences can be used to verify if the *actual data* follows a consistent behaviour.
An FSS can be used to formalise the domain business rules and to check the consistency of Finite State Event Datasets. Although the whole content of a database could be checked by an FSS, it is advisable to split the database in several subsets (each being a separate FSED) and to check each of them separately.

Definition 3.3 (Finite State Event Database) *Let S_i be an FSED, we define a Finite State Event Database (FSEDB) as a database DB whose content is DB = $\bigcup_{i=1}^{k} S_i$ where $k \geq 1$.*

How can an actual database be verified by a model checker? A schematic representation of this approach is depicted in Fig. 1(b):

1. A domain expert codifies the evolution of the system as well as the consistency properties using the model checking tool language (i.e., the model).
2. An FSED S_i is retrieved from the database (i.e., the FSEDB) and the model checker automatically generates an FSS representing the evolution of the model caused by S_i.
3. The model checker looks for an error trace on the FSS. A solution (if any) represents an inconsistency affecting the database event sequence S_i. Otherwise the event sequence is consistent.

Any model checker can be used to perform the verification. In our case, we used the CMurphi tool [7] which allows one to use C/C++ functions to interact with the database.

The concept of Consistency Failure Point (CFP) is now introduced: a CFP is an event of a FSED from which the sequence becomes inconsistent. The CFP event is not necessarily the responsible of the consistency failure, but it is the point where the failure emerges. The FSED is labelled as inconsistent if a CFP is discovered. The remaining of the event sequence can be hardly tested (or cannot be test at all) since the inconsistency might hinder the FSS-state-evolution identification thereafter. Considering the example of Tab. 1, the missing Lisbon departure prevents the exploitation of the FSS (Fig.1(a)) straight after the Lisbon checkin (the subsequent Barcelona checkin is the CFP), since other events could be missing, not only the Lisbon checkout. Generally speaking, the uncertainty originating after a CFP can prevent the execution of the consistency check for some or all the subsequent events. Considering again the example of Tab. 1, the uncertainty doesn't last for long time: the Barcelona harbour checkin event is enough to guess the FSS state and to resume the consistency check. In other cases, e.g. the one presented in Sec. 5, the uncertainty can last longer. The question is how to detect the points where the consistency check can be safely resumed. For this reason we introduce the *reset actions*. A *reset action* is an action so that the FSS state can be determined with certainty thereafter, even though the previous history is unknown. It can be observed that a reset action leads the FSS always to the same state, independently of the previous history. More formally:

Definition 3.4 (*Reset Action*) *Let S (S,I,A,F) be a Finite State System according to Def. 3.1, an action* $a \in A$ *is a reset action iff* $\exists s_a \in S$ *s.t.* $\forall s \in S$ *either* $F(s,a)=s_a$ *or* $F(s,a)$ *is not defined.*

Since events can be mapped to actions, the *reset event* can be defined in a similar way: it is the event that lead the FSS always to the same state, independently of the previous history. The reset events can be used for partitioning a dataset into small event segments whose consistency can be evaluated independently. An example is showed in Sec. 5. In this way, a CFP found within a segment does not prevent the consistency evaluation of the subsequent segments.

Running Example. The following example should clarify the matter. Let us consider the Cruise Ship example as introduced in Tab. 1.
The whole dataset is the FSEDB whilst a FSED is the travel plan of a single ship. An *event* e_i is composed by attributes *ShipID*, *City*, *Date*, and *Event Type*, namely e_i = (*ShipID*$_i$, *City*$_i$, *Date*$_i$, *Event*$_i$ *Type*$_i$). Moreover, the total-order operator ~ could be the binary operator $\leq$ defined over the event's attribute *Date*, hence $\forall e_i,e_j \in E$, $e_i \leq e_j$ iff $Date_{ei} \leq Date_{ej}$. Finally, a simply consistency property could be *"if a ship checks in to the harbour A, then it must check out from A before checking in to the next harbour"*. We can model this consistency property as a model checking problem. An implicit representation of the domain is given in Fig. 1(a). 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 is harboured. The consistency property of a database events sequence, e.g., the travel plan of Tab. 1, can be expressed as a model checking problem. In such a case, a solution (i.e., the error trace) is represented by the event sequence e_1, e_2, e_3, e_4 which generates an inconsistent trajectory on the corresponding FSS.

4 DATA CLEANSING VIA MODEL CHECKING

In the previous sections we described how the consistency of a database event sequence can be modelled and verified through model checking. Looking forward, one can wonder if the consistency model can be used as the basis to identify cleansing activities. Namely, once the FSS describing the dataset consistency is generated, can the FSS be exploited to identify the corrective actions that can make such dataset consistent? Let us consider an inconsistent event sequence having an action ai that applied on a (reachable) state si leads to an inconsistent state sj. Intuitively, a corrective action sequence represents an alternative route leading the system from state si to a state when the action ai can be applied (without violating the consistency rules). In other words, a *cleansing action sequence* (if any) is a sequence of actions that, starting from s_i, makes the system able to reach a new state on which the action a_i can be applied and results in a consistent state. More formally we can define the following.

Definition 4.1 (Cleansing Action Sequence) *Let S = (S,I,A,F) be an FSS, E be the set of errors states (i.e. inconsistent states) and T be the finite horizon. Moreover,*

- Let $\Omega = \bigcup_{i_i \in I} \text{Reach}(i_i)$ be the set of all the states reachable from the initial ones;
- Let $\pi = s_0 a_0 \ldots s_i a_i s_j$ be an *inconsistent trajectory* where $s_j \in \Omega$ is an inconsistent state (i.e., $s_j \in E$ and and $S_0, \ldots, S_i \notin E$.

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

In the AI Planning field a *Universal Plan* is a set of policy, computed off-line, able to bring the system to the goal from any feasible state (the reader can see [13, 5, 9] for details). Similarly, we are interested in the synthesis of an object, which we call *Universal Cleanser* (UC), which summarises for each *pair* (state, action) leading to an inconsistent state, the set *A'* of *all* the feasible cleansing action sequences. This UC is computed only once and then applied as an oracle to cleanse any kind of FSEDB.
To this aim, we proceed as follows:

Step 1 A consistency model of the system is formalised by means of a model checking language as described in Sec. 3.

Step 2 A database domain model is formalised, describing the attribute domains from which all the possible record subsets (i.e. event subsequences) composed by at most T events can be guessed (both the consistent an the inconsistent ones). The set of possible subsets will be called *worst case FSEDB* hereafter. E.g., for the cruise ship example an extract of the model is: $city=\{City_x, City_y\}$ $EType_i$={checkin,checkout}. Note that the City attribute cardinality (although potentially unbounded) can be limited by a finite and small number thanks to the number of state variables and to the FSS *diameter*[10].

Step3 The model checker is used to generate the FSS representing all the inconsistent sequences, starting from the database domain model (step 2) and the consistency model (step 1), the whole process is shown in Fig. 1(b).

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

Step4 Explore the FSS to synthesise the Universal Cleanser.

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

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

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

It is worth to highlight that, while on the one hand the UC generated is *domain-dependent*, i.e. it can deal only with event sequences conforming to the model that generated it, on the other hand it is *data-independent* since, once the UC is computed on a worst-case FSEDB, it can be used to cleanse *any* FSEDB. The pseudo code of the algorithm generating a Universal Cleanser is given in Procedures 1 and 2. It has been implemented on top of the UPMurphi tool [8]. The Procedure 1 takes as input the FSS of the domain, the set of error states given by the model checker (to identify inconsistent trajectories) and a finite horizon T. Then, it looks for a cleansing action sequence (according to Def. 4.1) for each inconsistent (state, action) pair. This work is recursively accomplished by the Procedure 2 which explores the FSS through a Depth First visit collecting and returning all the cleansing solutions.

Running Example. Consider again the Cruise Ship example of Tab. 1. We recall that an *event* e_i is e_i=(ShipID$_i$, City$_i$, Date$_i$, EType$_i$) and each event sequence and subsequence is ordered with respect to the event dates. It is worth to note that the finite horizon $T = 2$ is enough to guarantee that any kind of inconsistency will be generated and then corrected using no more than 2 actions. Note that the cardinality of the city attribute can be potentially unbounded, but since a state can store only one city information at a time, we can use two elements ($City_x$ and $City_y$) to represent any feasible $City_i$ value in the system. Consider that the main elements of an event are $EType_i \in \{checkin, checkout\}$, $City_i \in \{City_x, City_y\}$, i.e., 4 possible events. Then, we represent the *wost-case* FSEDB by considering into our model all the possible 2-step event subsequences (i.e., simply enrich each node of the graph in Fig. 1(a) with all the possible edges).
Table 2 shows the Universal Cleansing for our example, which is *minimal* with respect to the number of event variable assignments, i.e., the missing pair $([pos=sea], (checkout, City_y))$ fits on $([pos=sea], (checkout, City_x))$. The UC, once generated, is able to cleanse any kind of *FSEDB* compliant with the model from which it has been generated.

([state],(action))	list of corrective actions
([pos=sea], (checkout, City$_x$))	(checkin, City$_x$)
([pos=harbour ∧ city=City$_x$], (checkout, City$_y$))	(checkout, City$_x$),(checkin, City$_y$)
([pos=harbour∧city=City$_X$], (checkin,City$_Y$))	(checkout, City$_x$)
([pos=harbour∧city=City$_X$], (checkin,City$_X$))	(checkout, City$_x$)

Table 2: Universal Cleanser for the Cruise Ship Example.

Procedure 1 UNIVERSALCLEANSING

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

Procedure 2 AUXUC

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

5 THE CASE OF "THE WORKERS CAREER ADMINISTRATIVE ARCHIVE"

The Italian Law No. 264 of 1949 requires the employers to notify the public administration whenever an employee is hired, dismissed, or her/his working contract is modified. Those notifications are called *Mandatory Communications* ("Comunicazioni Obbligatorie" in Italian). Since the 1997, the Ministry developed an ICT infrastructure, called the *"CO System"* [16], for recording data concerning mandatory communications, employment, and active labour market policies. Some administrative archives useful for studying the labour market dynamics [11] are generated and called *"CO Archives"*or *"Job Registries*". Extracting the longitudinal data by the CO archives allows one to observe the overall *flow* of the labour market for a given observation period, obtaining insightful information about worker career paths, patterns and trends, facilitating the decision making processes of civil servants and policy makers [10].
Unfortunately the archive quality is very low, therefore cleansing is required before deriving information for decision making purposes (see, e.g. [3]). The approach presented in this paper has been used to perform data consistency evaluation and cleansing on the real data extracted from the CO archive of an Italian Area.

5.1 Domain Modelling

This subsection will provide some domain knowledge useful to achieve an overview of the administrative archives analysed in this paper. Every time an employer hires or dismisses an employee, or an employment contract is modified (e.g. from part-time to full-time, or from fixed-term to unlimited-term), a Mandatory Communication is notified to the CO System and stored into a job registry. The registries are managed at "*provincial level"* for several administrative tasks, every Italian province has its own job registry recording the working history of its inhabitants (as a side effect).
For each worker, a mandatory notification (an *event* in our context) is composed by:

w_id: it represents an id identifying the person involved in the event;
e_id: it represents an id identifying the communication;
e_date: it is the event occurrence date;
e_type: it describes the event type occurring to the worker career. The allowed event types are: the *start* or the *cessation* of a working contract, the *extension* of a fixed-term contract, or a contract type *conversion*;
c_flag: it states whether the event is related to a full-time or a part-time contract;
c_type: it describes the contract type with respect to the Italian law (e.g. fixed-term or unlimited-term contract, etc.).
empr_id: it uniquely identifies the employer involved in the event.

The evolution of a consistent worker's career along the time is described by a *sequence* of events ordered with respect to *e_date*. More precisely, in this settings an FSED is the ordered set of events for a given *w_id*, and the FSEDs union composes the FSEDB. Moreover, the representative element is given by the *w_id*. Now we closely look to the consistency of the worker careers, where the consistency semantics is derived from the Italian labour law, from the domain knowledge, and from the common practice. Some rules can be identified:

c1: an employee can have no more than one full-time contract active at the same time;
c2: an employee cannot have more than K part-time contracts (signed by different employers); in our context we assume K = 2 i.e., employees cannot have more than two part time jobs active at the same time;
c3: a contract extension cannot change neither the existing contract type (*c_type*) nor 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;
c4: a conversion requires either the *c_type* or the *c_flag* to be changed (or both).

For simplicity, we omit to describe some trivial constraints e.g., an employee cannot have a *cessation* event for a company for which she/he does not work, an event cannot be recorded twice, etc.
The CMurphi model checker allows us to build an FSS which will be used to check the data consistency. The system state (i.e., a worker's career at a given time point) is composed by three elements: the list of companies for which the worker has an active contract (*C[]*), the list of modalities (part-time, full-time) for each contract (*M[]*) and the list of contract types (*T[]*).
To give an example, *C[0]=12*, *M[0]=PT*, *T[0]=unlimited* models a worker having an active unlimited part-time contract with company 12.
The CMurphi model of the domain is showed in Figure 5.1 and it outlines a consistent career evolution. Note that, to improve readability, we omit to represent *conversion* events as well as inconsistent states/transitions (e.g., a worker activating two full-time contracts), which are handled by the FSS generated by the CMurphi model.
A valid career can evolve signing a part-time contract with company *i*, then activating a second part-time contract with company *j*, then closing the second part-time and then reactivating the latter again (i.e., *unemp*, emp_i, $emp_{i,j}$, emp_i, $emp_{i,j}$).

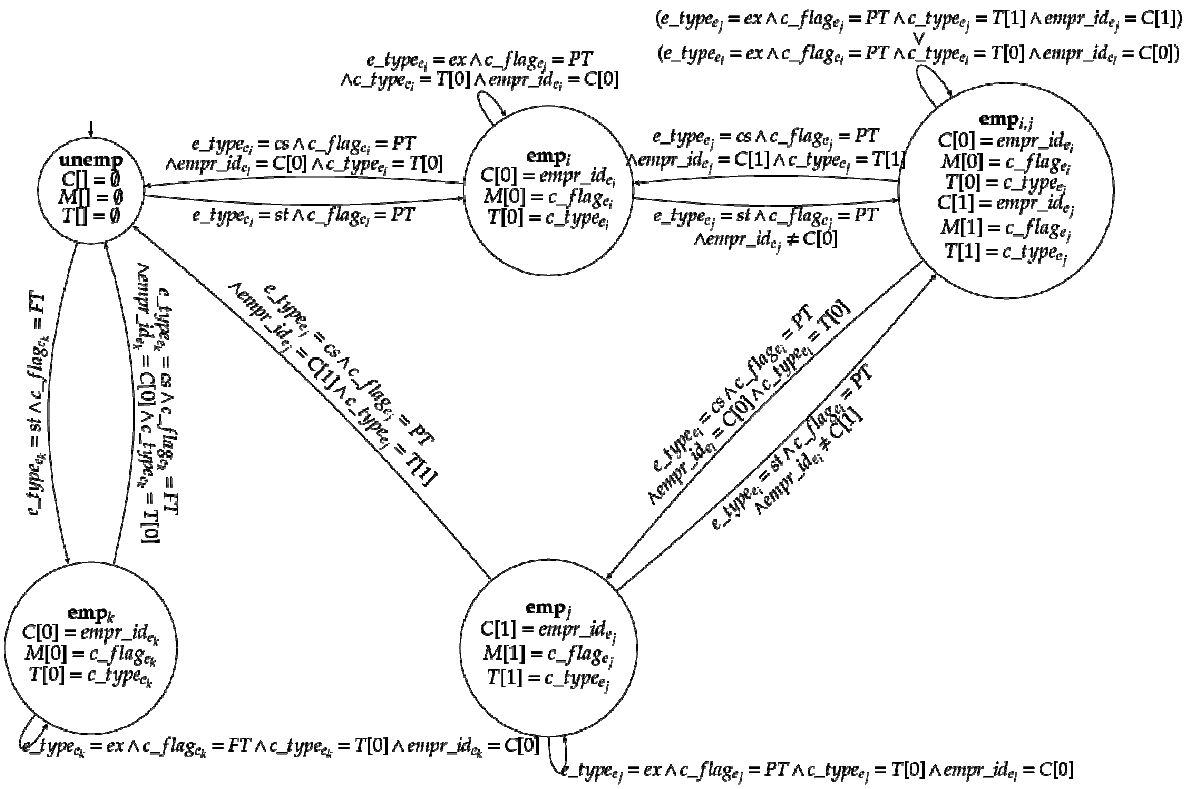


Figure 2: A graphical representation of an FSS of a valid worker's career where *st=start, cs=cessation, cn=conversion,* and *ex=extension.*

5.2 Data Consistency Experimental Results

We performed the consistency check using the model described in Fig. 5.1 on the "CO archive" of an Italian Area, composed by 1,248,751 mandatory communications. The CO archive (*S* from now on) describes how the labour market has evolved from the 1st January 2000 to the 31st December 2010, by providing CO events for 214,418 people careers. Each career has been modelled as a subset S_i where i∈ [1,…, 214,418]. An S_i is a *FSED* while *S* is the *FSEDB* according to the terminology introduced in the previous section.

The consistency check computation was performed on a 32 bits 2.2Ghz CPU (connected to a MySQL server through ODBC driver) in about 20 minutes using about 50 MB of RAM.

Our results show that the 43.2% of the careers are inconsistent. More precisely, the 43.2% have *at least* one inconsistency (i.e., a CFP has been found). On the contrary, only the 56.8% of the total careers have proved to be consistent. Clearly, once an inconsistency is detected at a given time point, the remaining part of the career cannot be further evaluated since the CFP may have unpredictable effects on the consistency of the remaining part. To mitigate this effect, we exploited the consistency model of Fig. 5.1 to discover *reset events* (according to Def. 3.4) and to partition the careers into smaller segments. The following example should help to better clarify the usefulness of the reset events. Let us consider a worker career extracted from the dataset, as presented in Tab. 3(a). According to the record having e_id=4, the worker w1 starts a new full-time contract in date 39504 without closing the on-going part-time. Due to this inconsistency, the whole career will be considered *inconsistent*, although only the first four events have been evaluated.

Focusing on the system described in Fig. 5.1, it can be observed that some events always lead the system

to a specific state regardless of the previous ones: e.g., looking at Fig. 5.1, a full time cessation always leads to the *unemp* state as well as a full time start always leads to the emp_k state. Indeed, in such cases, the state reached by the system can be guessed in spite of the previous uncertainty. These events contributing to reduce the uncertainty are the *reset events*.

w_id	id	e_date	e_t	c_flag	c_type	em_id
w1	1	38402	st	part-time	limited	$empr_1$
w1	2	38679	st	part-time	unlimited	$empr_2$
w1	3	39023	cs	part-time	limited	$empr_1$
w1	4	39504	st	full-time	unlimited	$empr_3$
w1	5	39651	cs	full-time	unlimited	$empr_3$
w1	6	39700	st	part-time	unlimited	$empr_4$
w1	7	40407	cs	full-time	unlimited	$empr_4$
w1	8	40632	st	full-time	limited	$empr_5$
w1	9	41449	ex	full-time	unlimited	$empr_5$
w1	10	41513	cs	full-time	limited	$empr_5$
w1	11	41726	st	full-time	limited	$empr_6$
w1	12	42089	ex	full-time	limited	$empr_6$

Table 3: (a) An example of a worker career (the data is not real although plausible).

Seg	w_id	id	e_date	e_t	c_flag	c_type	em_id
S1	$w1_1$	1	38402	st	part-time	limited	$empr_1$
	$w1_1$	2	38679	st	part-time	unlimited	$empr_2$
	$w1_1$	3	39023	cs	part-time	limited	$empr_1$
	$w1_1$	4	39504	st	full-time	unlimited	$empr_3$
S2	$w1_2$	4	39504	st	full-time	unlimited	$empr_3$
	$w1_2$	5	39651	cs	full-time	unlimited	$empr_3$
S3	$w1_3$	5	39651	cs	full-time	unlimited	$empr_3$
	$w1_3$	6	39700	st	part-time	unlimited	$empr_4$
	$w1_3$	7	40407	cs	full-time	unlimited	$empr_4$
S4	$w1_4$	7	40407	cs	full-time	unlimited	$empr_4$
	$w1_4$	8	40632	st	full-time	limited	$empr_5$
S5	$w1_5$	8	40632	st	full-time	limited	$empr_5$
	$w1_5$	9	41449	ex	full-time	unlimited	$empr_5$
S6	$w1_6$	9	41449	ex	full-time	unlimited	$empr_5$
	$w1_6$	10	41513	cs	full-time	limited	$empr_5$
S7	$w1_7$	10	41513	cs	full-time	limited	$empr_5$
	$w1_7$	11	41726	st	full-time	limited	$empr_6$
S8	$w1_8$	11	41726	st	full-time	limited	$empr_6$
	$w1_8$	12	42089	ex	full-time	limited	$empr_6$

Table 3: (b) The segmented career of (a).

Using the UPMurphi tool and the model described Fig. 5.1, we verified that the *full-time* events always lead to the same state, i.e. they are *reset events*.

Given a FSED (according to Def. 3.2) describing a career composed of the events e_1, e_2, … e_n the reset events e_{rej} (corresponding to full time events) are selected where $re_j \in [re_1, re_2, re_3, \ldots, re_k] \subseteq [e_1, \ldots e_n]$. The career can be splitted into segments as follows: $[e_1, e_{re_1}]$, $[e_{re_1}, e_{re_2}]$, $[e_{re_2}, e_{re_3}]$,…,$[e_{re_n}, e_n]$. Excluding the last event of each segment (which is repeated as first event of the following one), the segments are non overlapping. The last event repetition is required to carry out the segment consistency check. The FSS for verifying the segment consistency has been modified by taking into account that a career segment can start from several states, not only from the *unemp* one.
Considering the example of Tab. 3(a), the career is decomposed by creating 8 segments which can be now analysed independently, as showed in Tab. 3(b). The consistency analysis on the segments shows that *S1*, *S5*, and *S6* are inconsistent, whilst the remaining segments are consistent. *S1* is inconsistent because the job with employer $empr_2$ is not closed before the beginning of the full-time contract with $empr_3$, *S5* is inconsistent because the first extension event (*e_id=9*) has *c_type=unlimited* and the extensions of an unlimited contract is not allowed. In *S6* there is a *c_type* mismatch. As shown by this example, the segments can now be evaluated after the first inconsistency using the career segmentation.

We applied this approach on our administrative archive *S*, generating an new archive S^{segm} where each career has been decomposed into segments by using the reset events previously introduced. The consistency check has been used to evaluate the segments consistency. The results (and a comparison with the whole career results) are shown in Tab. 4. We highlight that the database *S* is largely composed by reset events (the full time events are about the 81% of total events) motivating the big dimension of the S^{segm} archive in terms of segments. For this reason, in S^{segm} a segment is now composed by a low average number of events, less than 2 per segment (not considering the duplicates). The number of consistent segments is the 78.3% compared to the 56.8% of the consistent careers (analysed as single entities). Thanks to the use of the reset events we obtained a more precisely evaluation of the consistency of *S* in terms of segments. Similarly, looking at the number of events belonging to inconsistent careers, the results show that now only the 28.3% of the total events of *S* belong to inconsistent segments (rather than the previous 72.2%).

Row	Dataset Analysis	*S* (careers)	S^{segm} (segments)
1	# Events	1,248,751	2,091,507
2	# Elements	214,418	1,057,090
3	#Consistent Elements	121,853 (56.8%)	828,194 (78.3%)
4	#Inconsistent Elements	92,565 (43.2%)	228,896 (21.7%)
5	#Events member of Consistent Elements	346,553 (27.8%)	895,906 (71.7%)
6	#Events member of Inconsistent Elements	902,198 (72.2%)	352,845 (28.3%)

Table 4: A comparison between careers and segments data

Even tough the use of the reset actions has showed a more limited impact of inconsistencies in *S*, the analysis confirms that the original database has a low quality, motivating the need for data cleansing.
The discussion about the reasons of such poor data quality is out of the scope of this paper, nevertheless it is mainly related to the data collection process (few controls, a lot of manual data entry especially before the 2005) and to some trivial errors (e.g. double entries) that can easily make the careers inconsistent.

5.3 Data Cleansing Experimental Results

We generated the Universal Cleanser using the model described in Fig. 4. We generated the FSS from the *worst-case* database by choosing a $T = 5$ finite horizon, which is high enough to guarantee that any reachable inconsistent state can be considered. Then, Procedures 1 and 2 have taken as input the FSS

generated and the error states *E*, to detect inconsistent trajectories. Finally, Procedures 1 and 2 have been used to synthesise the Universal Cleanser. The UC contains 288 different (state, action) pairs able to make consistent any FSEDB (conforming to the model) in no more than 3 steps, avoiding looping corrective actions. We observed that $T = 3$ is enough to guarantee that any inconsistency will be corrected, whilst using $T = 2$ some errors cannot be fixed. To give an example, let us consider an inconsistent trajectory (i.e., a career in such a case) in which the last consistent state is emp_{ij} with (M:[PT,PT], T:[Limited,Limited], C:[$Company_x$,$Company_y$]), then a cessation for a full-time contract with a new company arrives (i.e, an event as (cs, FT, Limited, $Company_z$)). In such a case, the UC suggests to choose between two corrective interventions (similar to each other) composed by 3 actions for each. The first intervention is: to close the first part-time contract, i.e. (cessation, PT, Limited, $Company_x$) then to close the second one (cessation, PT, Limited, $Company_y$) and finally to start the full-time contract according to the event received (start, FT, Limited, $Company_z$). The second intervention can be obtained by switching the first two cessation events.

We applied the UC generated to the dataset *S* to cleanse the inconsistent careers as follows. For each career S_i, when an inconsistency is found: (1) Let *inc* be a CFP (i.e. an inconsistency at a given sequence point) for the career S_i. (2) Look at the UC evaluating all corrective action sequences able to fix *inc*. (3) Select a suitable corrective action sequence (according to a given policy) and apply it. (4) Evaluate again the consistency of S_i. (5) Repeat steps 1-4 until no CFPs for the career S_i emerges.

In this work we focus on the UC synthesis. Investigating how to select corrective actions from the ones proposed by the UC is outside the scope of this paper. Nevertheless, for the sake of completeness, we detail how the UC has been used to cleanse the worker career archive. We implemented the step 2 by always selecting the corrective action sequence minimising (maximising) the (per worker) *average working days* indicator. Hence, we obtained two cleansed version of *S*, namely S^{min} and S^{max}, representing the cleansed versions of *S* in which inconsistent careers have been cleansed by minimising and maximising their working days respectively. In our settings, these distinct datasets allow us to perform a *sensitivity analysis* on the "working day" indicator with respect to the uncertainty due to inconsistencies. Clearly, once the UC is generated, the user can use any kind of policy for choosing a corrective action sequence. Finally, the complete UC has been made available at [1].

6 CONCLUSION AND FUTURE WORKS

In this paper we have shown how (longitudinal data) consistency verification tasks can be modelled as model checking problems, then we used the CMurphi verifier on some administrative archives to detect the inconsistent data. The analysed archives store the working histories of people living in an Italian area. An anonymous version of the archives has been used, according to the current law and privacy requirements. The results showed that the data quality of the source archives is very low: only about the 56% of people careers are consistent. To further investigate these results, we exploited the consistency model to partition the careers into small segments whose consistency can be analysed independently, obtaining a very fine grained evaluation of the data quality: the 78% of the segments turned out to be consistent.

Finally, we provided an algorithm working on the consistency model that can automatically build a *universal cleanser*: a cleanser *domain-dependent* (i.e., it focuses on the consistency of a specific domain) but *data-independent* (i.e., it can cleanse any kind of dataset compliant with the model).
Using model checking to evaluate a consistency model against actual data put into the hands of domain experts a powerful instrument contributing to a better comprehension of the domain aspects, of the data peculiarities, and of the cleansing issues.

As a future work we would like to explore the temporal logic to express consistency rules. Currently our research goes into the direction of comparing the universal cleanser with other approaches.

REFERENCES

[1] The universal cleanser of the worker career administrative archive. Public available at http://goo.gl/OH74F, 2012.

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

[3] M. Cesarini, M. Mezzanzanica, and M. Fugini. Analysis-sensitive conversion of administrative data into statistical information systems. *Journal of Cases on Information Technology*, 9(4):57–81, 2007.

[4] K. T. Cheng and A. S. Krishnakumar. Automatic functional test generation using the extended finite state machine model. In *Proceedings of DAC*, pages 86–91. ACM, 1993.

[5] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of AAAI/IAAI*, pages 875–881, 1998.

[6] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[7] CMurphi Web Page. http://www.dsi.uniroma1.it/ tronci/cached.murphi.html, 2011.

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

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

[10] P. Lovaglio and M. Mezzanzanica. Classification of longitudinal career paths. *Quality & Quantity*, pages 1–20, 2012. 10.1007/s11135-011-9578-y.

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

[12] T. C. Redman. The impact of poor data quality on the typical enterprise. *Commun. ACM*, 41:79–82, 1998.

[13] M. Schoppers. Universal plans of reactive robots in unpredictable environments. In *Proc. IJCAI*, 1987.

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

[15] S. Tee, P. Bowen, P. Doyle, and F. Rohde. Data quality initiatives: striving for continuous improvements. *International Journal of Information Quality*, 1(4):347–367, 2007.

[16] The Italian Ministry of Labour and Welfare. Annual report about the CO system, available at http://www.cliclavoro.gov.it/news/Documents/Rapporto_Annuale_Comunicazioni_Obbligatorie/executive_summary.pdf, 2012.