Pattern Functional Dependencies for Data Cleaning

Abdulhakim Qahtan[*] Nan Tang[♠] Mourad Ouzzani[♠] Yang Cao[♣] Michael Stonebraker[◇]
[*]Utrecht University [♠]Qatar Computing Research Institute [♣]University of Edinburgh [◇]MIT CSAIL
a.a.a.qahtan@uu.nl {ntang, mouzzani}@hbku.edu.qa
yang.cao@ed.ac.uk stonebraker@csail.mit.edu

Introduction: Functional dependencies (FDs) and their different variants, *e.g.,* conditional functional dependencies (CFDs), have been widely used in data cleaning and other data management tasks such as query optimization and data modeling. In addition, *patterns* (or regex-based expressions) are widely used to specify the format of a set of values in a given domain, *e.g.,* a Year column should contain *only* four digits. Nevertheless, all previous *integrity constraints* (ICs), including FDs and CFDs, are limited to work on the entire attribute values and do not exploit the intrinsic knowledge carried out by partial attribute values in the form of patterns.

We introduce ***pattern functional dependencies*** (PFDs), a new type of ICs that combines dependency- and regex-based theories. Note that, besides using PFDs to detect data errors that are hard to capture using existing methods, a positive side-effect is that PFDs can also serve as meta-knowledge to facilitate other data analytics tasks.

Error Detection with Traditional ICs: Consider two tables: D_1 with the schema (name, gender) in Table 1, and D_2 over the schema (zip, city) in Table 2, respectively.

	name	gender
r_1:	John Charles	M
r_2:	John Bosco	M
r_3:	Susan Orlean	F
r_4:	Susan Boyle	M
		F

Table 1: D_1: Name

	zip	city
s_1:	90001	Los Angeles
s_2:	90002	Los Angeles
s_3:	90003	Los Angeles
s_4:	90004	New York
		Los Angeles

Table 2: D_2: Zip

Erroneous cells, r_4[gender] in D_1 and s_4[city] in D_2, are annotated in pink. Their correct values, F and Los Angeles, are shown and highlighted in green. Suppose the following FDs are defined on these tables:

$$\varphi_1 : \textbf{Name}\ ([\mathsf{name}] \rightarrow [\mathsf{gender}])$$
$$\varphi_2 : \textbf{Zip}\ ([\mathsf{zip}] \rightarrow [\mathsf{city}])$$

FDs

where φ_1 states that name uniquely determines gender in table Name, and φ_2 says that zip uniquely determines city in table Zip. Clearly, φ_1 cannot detect the error r_4[gender] in D_1, because there is no other tuple r : (Susan Boyle, F) in D_1 – an FD requires two tuples to cause a violation. Similarly, φ_2 cannot detect the error s_4[city] in D_2.

One possible, but very expensive, way to detect errors in D_1 and D_2 is by using *many* constant CFDs, as shown below: where ϕ_1 means that in table Name, if someone's name is John Charles, then his gender value should be M. The other constant CFDs (ϕ_2–ϕ_8) can be interpreted similarly.

$$\phi_1 : \textbf{Name}\ ([\mathsf{name} = \mathsf{John\ Charles}] \rightarrow [\mathsf{gender} = \mathsf{M}])$$
$$\ldots$$
$$\phi_8 : \textbf{Zip}\ ([\mathsf{zip} = \mathsf{90004}] \rightarrow [\mathsf{city} = \mathsf{Los\ Angeles}])$$

CFDs

Key Observation: One *fundamental limitation* of previous ICs (such as FDs and CFDs)is that they enforce data dependencies using the entire attribute values. Consequently, they cannot specify the fine-grained semantics found in partial attribute values. A *key observation* is that by relaxing the limitation of previous FDs of operating on entire attribute values, we can specify a new type of dependencies that can capture partial attribute values that follow some regex-like patterns. For example, in D_1, the first name is enough to determine gender, *e.g.,* John is a male and Susan is a female; and in D_2, the first three digits of zip, *e.g.,* 900, are sufficient to determine the city Los Angeles. Let us now consider a new type of pattern-based constraints:

$$\lambda_1 : \textbf{Name}\ ([\mathsf{name} = \mathsf{John}\backslash\textvisiblespace\backslash\mathsf{A}*] \rightarrow [\mathsf{gender} = \mathsf{M}])$$
$$\lambda_2 : \textbf{Name}\ ([\mathsf{name} = \mathsf{Susan}\backslash\textvisiblespace\backslash\mathsf{A}*] \rightarrow [\mathsf{gender} = \mathsf{F}])$$
$$\lambda_3 : \textbf{Zip}\ ([\mathsf{zip} = \mathsf{900}\backslash\mathsf{D}\{2\}] \rightarrow [\mathsf{city} = \mathsf{Los\ Angeles}])$$

PFDs

where λ_1/λ_2 says that if someone's first name is John/Susan, then the gender is M/F ($\backslash$A$*$ matches any string; and λ_3 says that if a five-digit zip code starts by 900, then the city is Los Angeles ($\backslash$D{2} matches any two consecutive digits). Clearly, λ_2 can detect error r_4[gender] in D_1 and λ_3 can detect error s_4[city] in D_2.

Alternatively, consider two other constraints as follows:

$$\lambda_4 : \textbf{Name}\ ([\mathsf{name} = \overline{\backslash\mathsf{LU}\backslash\mathsf{LL}*\backslash\textvisiblespace}\ \backslash\mathsf{A}*] \rightarrow [\mathsf{gender}])$$
$$\lambda_5 : \textbf{Zip}\ ([\mathsf{zip} = \overline{\backslash\mathsf{D}\{3\}}\ \backslash\mathsf{D}\{2\}] \rightarrow [\mathsf{city}])$$

PFDs

where λ_4 says that one's first name uniquely determines one's gender for table Name (assuming that name is written as first name followed by last name) ($\backslash$LU matches any upper case letter and $\backslash$LL$*$ matches any consecutive lower case letters); and λ_5 states that the first 3 digits of a 5-digit zip code determines the city for table Zip. These two PFDs (λ_4 and λ_5) are defined over a pair of tuples, *e.g.,* two tuples match as specified by the left hand side (LHS) of λ_4 if they both satisfy the pattern $\backslash\mathsf{LU}\backslash\mathsf{LL}*\backslash\textvisiblespace\backslash\mathsf{A}*$, and their first names are the same, which is enforced by $\overline{\backslash\mathsf{LU}\backslash\mathsf{LL}*\backslash\textvisiblespace}$.

λ_4 can detect error r_4[gender] by comparing r_3 and r_4: they have the same first name Susan but different gender, which identifies a violation consisting of four cells (r_3[name], r_3[gender], r_4[name], r_4[gender]). Similarly, λ_5 can detect error s_4[city] by comparing s_4 with s_1, s_2, or s_3.

<u>*Remark.*</u> Specialized PFDs such as λ_1–λ_3 are more conservative, and more general PFDs such as λ_4–λ_5 are less conservative, potentially leading to false positives (*e.g.,* a unisex name cannot determine the gender). Also, and not surprisingly, real-world data is not homogeneous. Taking Boston as an example, the first three digits of a zip code in Boston could be either 201, 202, 203, or 204, not unique as in the case of Los Angeles.