"Sanity Checks" over Auto-Extracted Family-History Data

Scott N. Woodfield, David W. Embley, Stephen W. Liddle and Christopher Almquist

1 Introduction

Automated information-extraction systems (and sometimes even humans) can extract erroneous (even ridiculous) genealogical data. Mothers do not bear children before they are born, and it is highly unlikely that they bear children before age ten or eleven or after age fifty or sixty. Coding procedurally to recognize and report conflicting, erroneous, and unreasonable fact assertions is possible, but the declarative solution we proffer here requires less effort, is more modular and generalizable, and is more conceptually sound.

Section **??** describes the formal conceptual model that underlies the solution. Section **??** describes its application to detecting and reporting potentially erroneous fact assertions extracted automatically by an ensemble of automated tools.

2 Conceptualization

The declarative solution has an evidence-based conceptual model as its formal foundation.[1] Figure **??** shows an example—a conceptualization with its predicates, hard and soft constraints, and documenting evidence.

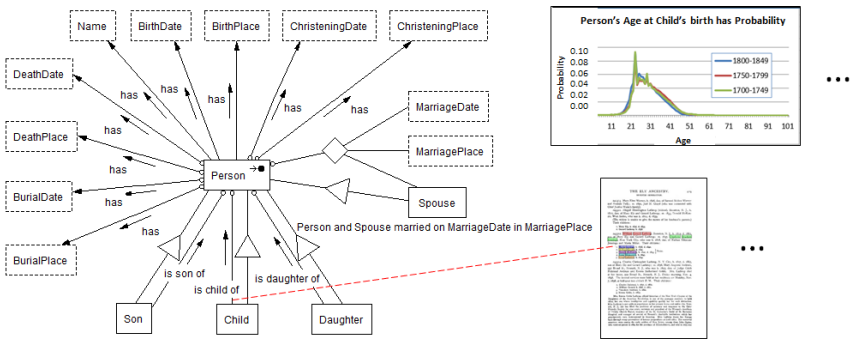

Figure 1: Depiction of Conceptual Model Features

Object sets, depicted as named rectangular boxes, are one-place predicates (e.g. $Person(x)$). Relationship sets, depicted by lines connecting object sets, are n-place predicates (e.g. $Person(x)$ *has* $BirthDate(y)$). Observe that predicates are in infix form and that predicate names come directly from the text and reading direction arrows in the diagram.

Constraints can be hard (returning only *satisfied* or *not satisfied* when checked) or soft (returning a *probability of being satisfied* when checked). The conceptual-model diagram in Figure **??** has 28 hard participation constraints specifying a minimum and maximum number of times an object may participate in a relationship set. Each object-set/relationship-set connection has one participation constraint as denoted by the decorations on the ends of the connecting lines. The diagram also shows 4 hard subset constraints (denoted by triangles on connecting lines) specifying that the objects in an object set must be a subset of the objects in another object set. In addition, Figure **??** shows one of many possible soft constraints as a probability distribution (*Person's Age at Child's birth has Probability*). It indicates, as well, that evidence can be associated with every predicate assertion instance (*Child is child of Person* statements found in a document).

[1] D.W. Embley, S.W. Liddle, S.N. Woodfield, "A Superstructure for Models of Quality," *Advances in Conceptual Modeling*, LNCS 8823, 2014, pp. 147–156

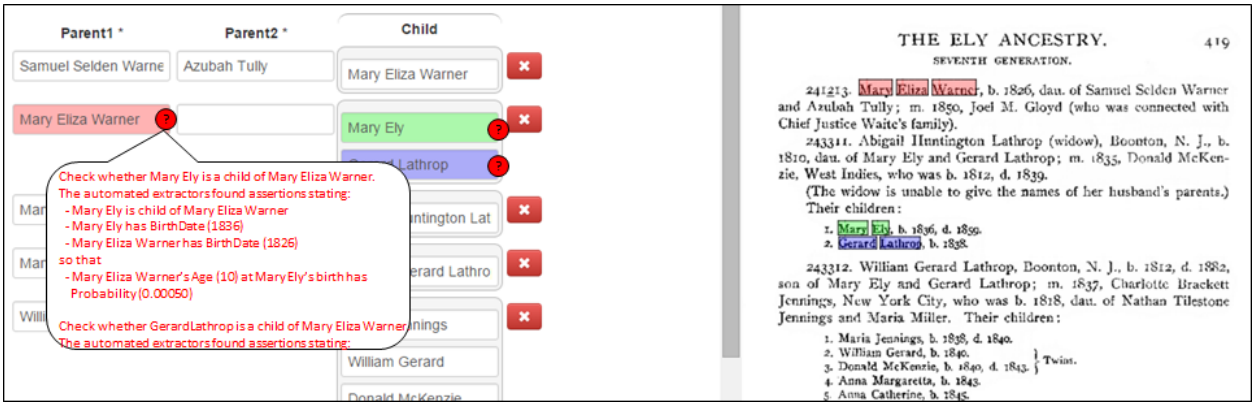

Figure 2: Extracted Data, Warning Markers, Messages, and Original Text.

Since the conceptual model is based on predicate-calculus, constraint rules can all be implications. The antecedents of an implication are predicates in the model or derived from these predicates or from given probability distributions, and the single consequent gives the probability of a condition being satisfied. For example, a rule for age difference between a parent and child is: $Child(x_1)$ *is child of* $Person(x_2)$, $Person(x_1)$ *has* $BirthDate(x_3)$, $Person(x_2)$ *has* $BirthDate(x_4)$, $Age(x_5) = Age(YearOf(x_3) - YearOf(x_4))$, *person's* $Age(x_5)$ *at child's birth has* $Probability(x_6)$ $\Rightarrow$ $Person(x_2)$*'s* $Age(x_5)$ *at* $Child(x_1)$*'s birth has* $Probability(x_6)$.

3 Application

Any probability that fails to meet a user-specified threshold is a constraint violation. Violations tell us that one or more of the antecedents must be incorrect. Thus, each asserted predicate instance should be checked—run through a "sanity check" with respect to the constraints.

Figure **??** shows a screenshot of a check request displayed for a user. The "evidence document"—the page from which the ensemble of tools extracted its assertions—is displayed on the right. The extracted information is in a sequence of filled-in record templates on the left. When a user hovers over a record, the filled-in fields are highlighted along with corresponding highlights of the text in the evidence document that the ensemble of tools extracted and placed in form fields. When a field has a check-request associated with it, an icon appears when the user hovers over the field. Clicking on the icon causes a pop-up to open revealing the message(s) that pertain to the field.

Each possible constraint violation has an application-dependent handler. Interestingly, the handler code can be generated automatically. Given the implication rule with its associated instance data and the user-chosen threshold for constraint violation, the handler fills in a template with the instance data found to be in violation. The handler generator substitutes textual instance values for variables in unary predicate-statement phrases (such as $BirthDate(x)$) and formats them for ease of reading. Since non-textual objects (such as *Person* instances and *Child* instances) come into existence by the principle of ontological commitment, the handler generator replaces unary person predicates with the person's name—the trigger for committing the ontology to recognize the existence of a person. The pop-up in Figure **??** shows the result for our example.

4 Concluding Remarks

A declarative constraint-violation checker can ease both administrator constraint specification and user adjudication. A prototype implementation of "sanity checks" in the context of an ensemble of automated information extractors is nearing completion.