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

Scott N. Woodfield[1], David W. Embley[1],
Stephen W. Liddle[2] and Christopher Almquist[1]

[1] Department of Computer Science
[2] Information Systems Department
Brigham Young University, Provo, Utah 84602, USA

Abstract. A declarative constraint-violation checker and message generator 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 illustrates its usefulness.

Keywords: automated information extraction, declarative constraint specification, checks for unreasonable genealogical fact assertions.

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 2 describes the formal conceptual model that underlies the solution. Section 3 describes its application to detecting and reporting potentially erroneous fact assertions extracted automatically by an ensemble of automated tools. Section 4 summarizes and makes concluding remarks.

2 Conceptualization

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

The diagram in Figure 1 graphically represents a logic database schema [2]. 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

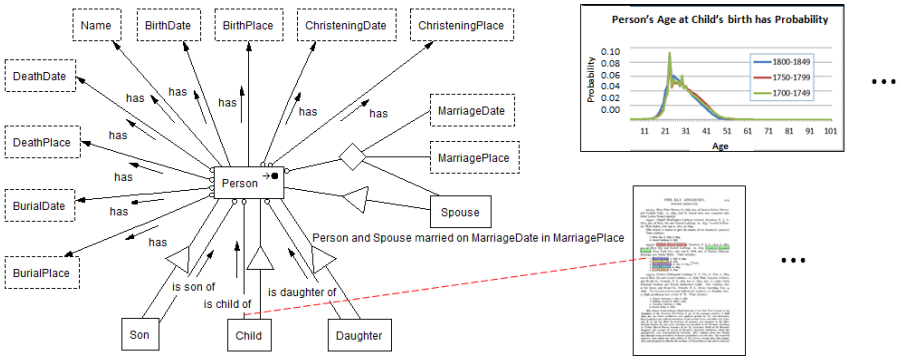

Fig. 1. Depiction of Conceptual Model Features

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 either *satisfied* or *not satisfied* when checked) or soft (returning a *probability of being satisfied* when checked). The conceptual-model diagram in Figure 1 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 1 shows one of many possible soft constraints as a probability distribution (*Person's Age at Child's birth has Probability*). The figure indicates, as well, that evidence can be associated (and in our automated-extraction application, is associated) with every predicate assertion instance (*Child is child of Person* statements found in a document).

3 Application

3.1 Hard Constraints

The conceptual-model diagram itself declaratively specifies hard cardinality constraints [3]. For example, it specifies that a person has one and only one birth date. The *Person* side of the *Person has BirthDate* relationship set has no "o" ("o" for "optional") on its connection and thus declares mandatory participation of a person object in the relationship set (at least one birth date). The *BirthDate* side of the relationship set has an arrowhead, which specifies that the relationship from *Person* to *BirthDate* is functional (at most one birth date).

Figure 2 shows an example of the proposed arbitrator interface with extracted data, warning markers, messages, and original text for the case of more than one birthdate having been extracted for Theodore Andruss. The generated message explains that one of the extraction tools correctly extracted "i860" as

Fig. 2. Two *BirthDate*s found for Theodore Andruss.

Theodore Andruss's birth date, which was changed to "1860" in a subsequent OCR-error-correction step. Because of the OCR error, however, another tool which expected date years to always have four digits found "1862" as the closest birthdate matching the pattern "b.\s\d4" following the name "Theodore Andruss".[3] In this case, the arbitrator should do nothing since the ensemble has already chosen the correct birth date for Theodore.

Interestingly, the declaration of a participation constraint is sufficient to generate code that both checks for participation constraint violations and handles them. In a populated model instance counting the number of times an object participates in a relationship set is straightforward as is checking whether the count is within a *min*–*max* range. Similarly, generating a handler that lists the violating objects in a statement template is also straightforward.

3.2 Soft Constraints

In our application soft constraints are based on probability distributions. Since the conceptual model is, in essence, predicate calculus, constraint rules can all be Datalog-like implications [4]. 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:

$$
\begin{aligned}
&\mathit{Child}(x_1)\ \mathit{is\ child\ of}\ \mathit{Person}(x_2),\\
&\mathit{Person}(x_1)\ \mathit{has}\ \mathit{BirthDate}(x_3),\\
&\mathit{Person}(x_2)\ \mathit{has}\ \mathit{BirthDate}(x_4),\\
&\mathit{Age}(x_5) = \mathit{Age}(\mathit{YearOf}(x_3) - \mathit{YearOf}(x_4)),\\
&\mathit{person's}\ \mathit{Age}(x_5)\ \mathit{at\ child's\ birth\ has\ Probability}(x_6)\\
&\Rightarrow\\
&\mathit{Person}(x_2)\mathit{'s}\ \mathit{Age}(x_5)\ \mathit{at}\ \mathit{Child}(x_1)\mathit{'s\ birth\ has\ Probability}(x_6).
\end{aligned}
$$

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.

[3] In the interface, hovering over a record highlights the information in the fields on both the form and the document from which the ensemble of extractors obtained the information. If a check-message is associated with a field in the record, hovering also causes a circled-question-mark icon to appear. The icon's color indicates the highest level of severity among the messages—green for an informational message, orange for a warning message, and red for an error message. Clicking on the icon pops open the associated messages.

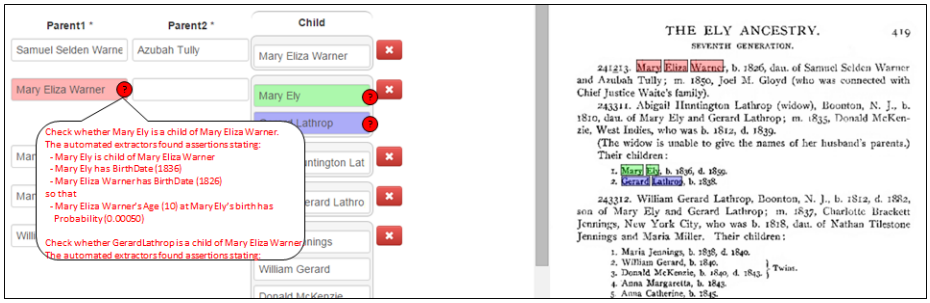

Fig. 3. Unreasonable Age Difference for Mary Eliza Warner and Her Children.

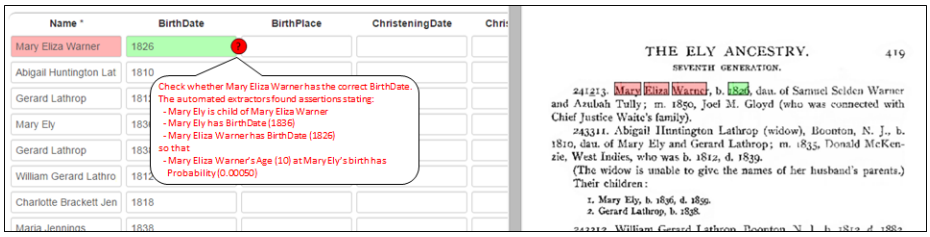

Fig. 4. Questionable BirthDate (Parent-Child Age Difference Unreasonable).

Thus, each asserted model predicate instance should be checked. Figures 3 and 4 show a check request displayed for a user.

Each possible constraint violation has an application-dependent handler. Interestingly, given only the Datalog, both the code to check the for a violation and code to handle a violation can be generated automatically. The checker code need only run its usual interpreter on the given the Datalog statement, which in essense creates a relational table in which each tuple is the join all predicate instances that satisfy the Datalog statement. These tuples are then fed one at a time to the handler. Given a user-chosen threshold for constraint violation, the handler fills in a message template with extracted 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-ups in Figures 3 and 4 show the result for our example. Note that since it is not known which of the antecedent predicates containing the extracted data is in error, the system must generate messages for all of them. Thus, similar to the message in Figure 4, a third message for the possibility of the child's *BirthDate* being in error is also generated and revealed when the user clicks on the warning icon associated with the child's birth date.

4 Summary and Concluding Remarks

The proposed system for doing "sanity checks" over asserted family-history information extracted automatically from historical documents has several desirable properties:

1. Code that checks for and handles any and all model-specified participation-constraint violations need only be written once (or can even be generated).
2. Adding a probability-distribution constraint requires only the writing an appropriate Datalog rule.
   (a) Code to check and handle any rule need only be written once (or can be generated).
   (b) Rules can be added, modified, and retracted dynamically.
   (c) With access to large genealogical data repositories such as those owned by FamilySearch [5], probability distributions for desired rule consequent statements are readily obtainable.
3. To the extent to which user-specified Datalog rules reflect reality, constraint checkers and handlers never error in identifying possible extraction errors. (They do not, however, know which one or more of the extracted fact assertions in its antecedent predicates are in error.)

These properties reduce the effort required of a system administrator who has the responsibility to code both constraint checkers and constraint-violation handlers. By pointing out possible errors at different levels of severity, adjudicators receive "just-in-time" messages for their task of arbitrating among conflicting assertions made by the various extraction tools in the ensemble and adding and retracting missed fact assertions.

References

1. D.W. Embley, S.W. Liddle, and S.N. Woodfield. A superstructure for models of quality. volume LNCS 8823, pages 147–156, 2014.
2. D.W. Embley and A. Zitzelberger. Theoretical foundations for enabling a web of knowledge. In *Proceedings of the Sixth International Symposium on Foundations of Information and Knowledge Systems (FoIKS'10)*, pages 211–229, Sophia, Bulgaria, February 2010.
3. S.W. Liddle, D.W. Embley, and S.N. Woodfield. Cardinality constraints in semantic data models. *Data & Knowledge Engineering*, 11(3):235–270, 1993.
4. H. Gallaire and J. Minker, editors. *Logic and Data Bases, Symposium on Logic and Data Bases*. Advances in Data Base Theory. Plenum Press, New York, 1978.
5. FamilySearch. http://familysearch.org.