Coq formalisation and mechanisation of privacy aware data integration

Thibaut Balabonski Véronique Benzaken Évelyne Contejean

International scientific positionning

Quoting [2]: "Modern hardware and software are monstrously complex. The best tool for coping with this complexity is *abstraction*—i.e., breaking up functionality into components or layers, with interfaces that are as narrow and clear as possible. Each interface is accompanied—implicitly or explicitly—by a specification expressing the contract between providers and consumers of that interface. These specifications come in a multitude of different forms: comments in code and natural-language documentation, unit tests, assertions, contracts, static types, property-based random test suites, and formal specifications in various logics. Sadly, despite widespread agreement on the importance of abstraction, specifications are often seen as an afterthought, or even a hindrance, to system development. Why? Experience has shown that it is extremely challenging to write good ones. Indeed, a maximally useful interface specification must be simultaneously rich (describing complex component behaviors in detail); two-sided (connected to both implementations and clients); formal (written in a mathematical notation with clear semantics to support tools such as type checkers, analysis and testing tools, automated or machine-assisted provers, and advanced IDEs); and live (connected via machine-checkable proofs to the implementation and client code). Specifications with all of these properties are called deep specifications." Obtaining such specifications requires the use of formal methods and mature tools. A very promising approach consists in using proof assistants such as Isabelle [9] or Coq [8]. In addition, and regardless to deep-specification efforts pursued by the formal methods community, current data management applications and systems have amazingly escaped to this unprecedented, world-wide effort. This is surpring as with the advent of Big data such systems and data engines store and manage and integrate increasingly massive data volumes. The Datacert project aims at providing deep-specification of privacy aware data integration. This intern is settled in this context[1].

Specific scientific context and motivation

As witnessed by a large number of strategic reports, white papers, studies it is now broadly recognized that in a wide range of application areas (business, society and science), data is produced, integrated and consumed at an unprecedented scale and this creates new theoretical and technological challenges for the IT (Information Technology) community. Data integration and sharing are often impeded by **privacy** concerns which has become a **key bottleneck** to deploying data integration platforms. This, undesiderable situation, is mainly due to the fact that **no data integration or data sharing algorithms and systems** do seriously take into account **privacy-by-design** nor do they provide **strong guarantees** that they really conform to their specifications and, hence, that, privacy is enforced.

We propose a radically different alternative to the majority of current data integration and sharing approaches in presence of security policies. Rather, we propose to develop a *deeply-specified privacy by design* approach where data integration algorithms are formally verified for correctness with regards to a given set of properties. This intern proposal is part of the ANR supported Datacert project, whose overall objective is to obtain a *Coq certified platform for privacy-aware data integration and exchange*. This includes providing deep specification of different data models, of various security and privacy policies, of most known or to be designed by the project (privacy preserving) integration and exchange algorithms to

[1] This intern is **not** a database oriented proposal.

date, formally proven correct translations into the world-wide database standard SQL, yielding a *complete suite of correct-by construction programs*.

Research program

Data integration and exchange aims at providing a unique *user entry point* to a set of data sources $D_1, \ldots, D_n$, also called *local data*. Users only need to access it, issuing a query Q, rather than querying each local data separately and then recombining the results. Exchange and integration are based on two slightly different mechanisms. On the one hand, the *data integration* system is in charge of identifying and accessing relevant sources by *rewriting* Q into a set of subqueries Q_i, collecting the partial results A_i and finally, combining them into a complete answer A which is returned to the user. On the other hand, *data exchange* materializes[2] an instance of the target schema that reflects the data in the sources[3]. In this setting, the user query Q is directly evaluated on the materialized (target) instance. Both data integration and data exchange rely on the ability of defining correspondences, called *schema mappings*, between the different source's schemas and the target schema. Formally, schema mappings are high-level assertions, typically expressed in first order logic, involving semantic correspondences between sources and target. Such assertions fall in the (well-known in the database dialect) class of *tuple generating dependencies (tgd's)* that are logical assertions of the shape

$$\forall \vec{x},\ \phi(\vec{x}) \implies (\exists \vec{z},\ \psi(\vec{x} \cup \vec{z})) \qquad \text{(tgd's or schema mapping)}$$

In their most general form, tgd's are called *Global-Local-as-View (GLAV)*, and ϕ and ψ are conjunctions of atoms. This terminology refers to the data integration framework, in which ϕ uses the sources' (*i.e.*, local) vocabulary, whereas ψ uses the target (*i.e.*, global) vocabulary. Well known restrictions are *Local-As-View (LAV)* and *Global As View (GAV)* tgd's. For the former, ϕ is a single atom, that is the sources are defined as views on the global schema, [5] while for the latter, ψ consists in a single atom. Tgd's can also be used for defining constraints on the target instance, and in that case both ϕ and ψ use the target vocabulary.

In this setting, one needs to be able to materialize a suitable target instance. The *chase algorithm* is a standard tool for producing it: applied to a set of tgd's, the chase algorithm produces the most general target. However, even in reference textbooks [1], the chase appears to be either under specified or even faulty as illustrated in [4]. Obviously, in such a setting, this could lead to disastrous consequences.

Therefore, tuple generating dependencies, various versions of the chase algorithm, translations from tgd's into SQL are at the heart of both data integration and security. A preliminary step consists in providing the necessary Coq formalisation for those. This intern, is a first step in this direction and leads to a **PhD funded research track** that will be organized in three distinct, though connected and intertwined, tracks.

1. *Coq formalisation of tuple generating dependencies:* based on the Coq implementation in [3], where only GAV dependencies were taken into account, we shall consider first LAV dependencies and then the (full extended tuple generating) GLAV dependencies.

2. *Coq mechanization of chase variants:* based on the preliminary formalisation of the chase given in [3], we shall further formalise existing variants of the chase algorithm for data exchange [6]. Then, the variants of the chase algorithms to be extracted are those that are more viable to implementation (such as, for instance, the oblivious chase and the semi-oblivious chase [6]). This formalisation will yield results such as, Coq proofs of correction and Ocaml extraction of all chase versions under consideration.

3. *Coq mechanized SQL semantics and translations:* based on a Coq mechanization of the SQL language currently under development, we shall address the Coq formalisation of the algorithm [7] for translating tgds into SQL insert statements. Such a tgd-to-SQL translation algorithm is the core of the Clio data exchange engine, developed by IBM. Clio was one of the first tools to systematically study the schema mapping problem. The research prototype is still maintained and refined at IBM, whereas the commercial version of the tool is part of the IBM InfoSphere Platform. This tool and

[2] computes and physically stores.

[3] Often in data exchange there is a unique source.

its underlying algorithms has inspired the research on data exchange during the last 15 years. We shall thus formalise this algorithm. As such, the goal is to obtain a Coq certified translation of this tgd-to-SQL translation algorithm and an Ocaml extraction of such translation.

Prerequisite

The candidate should have a strong background in theoretical computer science with emphasis in logic as well as a concrete practical experience of functional programming in OCaml-like programming languages. A **good** knowledge of Coq is mandatory.

Funding

This intern will be funded in the context of the ANR grant DATACERT. **A dedicated grant is provided for a three years PhD if the candidate demonstrates the expected skills to enroll in a PhD programme**.

Opportunities

In terms of academic skills, this internship will immerge the candidate in the realm of interactive theorem proving and offer her/him the opportunity to acquire or improve a first experience with Coq. In terms of professional skills, this internship will allow her/him to acquire a first experience with the emerging profession of "proof engineer" as witnessed by many job offers such as the one found at: `http://ssrg.nicta.com.au/jobs/proof-engineers2015`.

Location and duration

This intern will take place within the VALS research group of LRI lab in Paris Sud (bâtiment 650). The VALS group is a world-wide recognised research team in the area of Verification and Validation of Algorithms, Languages and Systems, right in the heart of the scientific field called "Formal Methods" (`https://vals.lri.fr/`). The applicant will be co-advised by Dr., Th. Balabonski, Prof., V. Benzaken and Dr., É. Contejean. The duration is of six months.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] A. Appel, A. Chlipala, B. Pierce, Z. Shao, S. Weirich, and S. Zdankevicz. The science of deep specification. NSF Expedition Project Proposal, personal communication.

[3] Véronique Benzaken and Évelyne Contejean. A Coq library for the relational model of data, 2012. `http://datacert.lri.fr/`.

[4] Véronique Benzaken, Évelyne Contejean, and Stefania Dumbrava. A Coq formalization of the relational data model. In Z. Shao, editor, *European Symposium on Programming, LNCS 8410*, pages 189–208, April 2014.

[5] Marc Friedman, Alon Y. Levy, and Todd D. Millstein. Navigational plans for data integration. In *AAAI/IAAI*, pages 67–73, 1999.

[6] Adrian Onet. The chase procedure and its applications in data exchange. In Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt, editors, *Data Exchange, Integration, and Streams*, volume 5 of *Dagstuhl Follow-Ups*, pages 1–37. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[7] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *VLDB*, pages 598–609, 2002.

[8] The Coq Development Team. *The Coq Proof Assistant Reference Manual*, 2010. `http://coq.inria.fr`.

[9] The Isabelle Development Team. *The Isabelle Interactive Theorem Prover*, 2010. `https://isabelle.in.tum.de/`.