Proving code equivalence in database-driven applications and SPARQL queries

Milena Vujošević Janičić

Department for Computer Science
Faculty of Mathematics, University of Belgrade, Serbia
www.matf.bg.ac.rs/~milena

Women in EuroProofNet
Nantes, June 24th, 2022.

About me and this this talk

- Associate professor at Department for Computer Science, Faculty of Mathematics, University of Belgrade, Serbia, www.matf.bg.ac.rs/~milena
- Principal researcher at Oracle labs (Switzerland)
- Research interests: Software Analysis, Optimization and Verification
- Joint work with Mirko Spasić
  - Mirko Spasić, Milena Vujošević Janičić.
    Solving the SPARQL Query Containment Problem with SpeCS.
    Submitted to Journal of Web Semantics.
  - Mirko Spasić, Milena Vujošević Janičić.
    Verification supported refactoring of embedded SQL.
    Software Quality Journal, Springer (2020)

Overview of the Presentation

Query Containment Problem

Definition

The query containment problem is a problem of deciding if each result of one query is also a result of another query (for any data-set).

- Fundamental problem in data management, introduced in **1977**
- **Undecidable** problem, **NP-complete** for conjunctive queries and unions of conjunctive queries
- Global query optimizations done by static query analysis
  - Query containment problem
  - Query equivalence
  - Query satisfiability

Semantic Web and SPARQL

- Semantic Web — making Internet data machine-readable
- Resource Description Framework (**RDF**)— a standard model for data interchange on the Web (data model for metadata, framework for defining relationships between data objects)
- Growing amount of data in RDF
- **SPARQL** is an RDF query language — enables users to query information from databases or any data source that can be mapped to RDF

RDF Example — N-Triples — S P O

```
<http://dbpedia.org/resource/Bob_Marley> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
<http://dbpedia.org/resource/Bob_Marley> <http://www.w3.org/2000/01/rdf-schema#label> "Bob Marley"@en .
<http://dbpedia.org/resource/Bob_Marley> <http://www.w3.org/2000/01/rdf-schema#label> "Bob Marley"@fr .
<http://dbpedia.org/resource/Bob_Marley> <http://www.w3.org/2000/01/rdf-schema#seeAlso> <http://dbpedia.org/resource/Rastafari> .
<http://dbpedia.org/resource/Bob_Marley> <http://dbpedia.org/ontology/birthPlace> <http://dbpedia.org/resource/Jamaica> .

<http://dbpedia.org/resource/Jamaica> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Country> .
<http://dbpedia.org/resource/Jamaica> <http://www.w3.org/2000/01/rdf-schema#label> "Jamaica"@en .
<http://dbpedia.org/resource/Jamaica> <http://www.w3.org/2000/01/rdf-schema#label> "Giamaica"@it .
<http://dbpedia.org/resource/Jamaica> <http://www.w3.org/2003/01/geo/wgs84_pos#lat> "17.9833"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://dbpedia.org/resource/Jamaica> <http://www.w3.org/2003/01/geo/wgs84_pos#long> "-76.8"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://dbpedia.org/resource/Jamaica> <http://xmlns.com/foaf/0.1/homepage> <http://jis.gov.jm/> .
```

RDF Example — Graphic representation

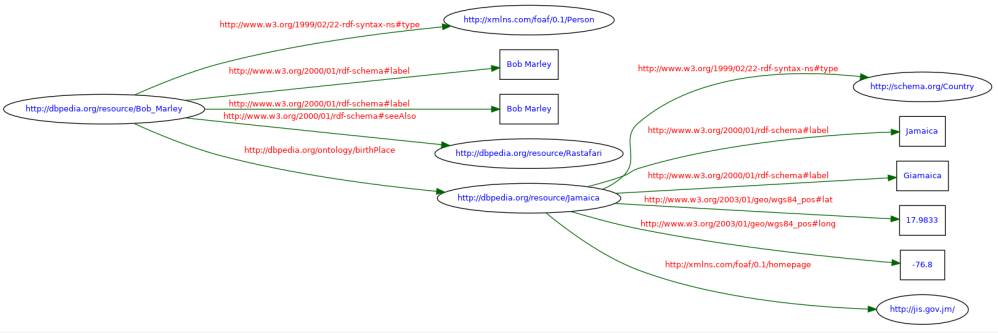


Figure: Graphic representation of RDF data.

Example of SPARQL queries

Extracting data about musicians from Jamaica

```
SELECT ?name ?surname ?bDay ?dDay {
    ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?t .
    ?t <http://www.w3.org/2000/01/rdf-schema#label> "musical artist"@en .
    ?s <http://xmlns.com/foaf/0.1/givenName> ?name .
    ?s <http://xmlns.com/foaf/0.1/surname> ?surname .
    ?s <http://dbpedia.org/ontology/birthDate> ?bDay .
    OPTIONAL { ?s <http://dbpedia.org/ontology/deathDate> ?dDay } .
    ?s <http://dbpedia.org/ontology/birthPlace> ?place .
    ?place <http://www.w3.org/2000/01/rdf-schema#label> "Jamaica"@en .
}
```

Result of a SPARQL query

Set of mappings:

{

{ ?name ↦ "Bob", ?surname ↦ "Marley", ?bDay ↦ "1945.02.06", ?dDay ↦ "1981.05.11" },

{ ?name ↦ "Arthur", ?surname ↦ "Reid", ?bDay ↦ "1915.07.21", ?dDay ↦ "1975.01.01" },

{ ?name ↦ "Jimmy", ?surname ↦ "Cliff", ?bDay ↦ "1948.04.01" },

. . .

}

SPARQL Syntax

```
Query    ::= 'select' Vars
             ('from' iri)*
             ('from' 'named' iri)*
             'where' '{' GPatt '}'
Vars     ::= '*' | var+
GPatt    ::= TPatt
           | GPatt '.' GPatt
           | GPatt 'union' GPatt
           | GPatt 'minus' GPatt
           | GPatt 'diff' GPatt
           | GPatt 'optional' GPatt
           | GPatt 'filter' Cond
           | '{' GPatt '}'
           | '{' SubQuery '}'
           | 'graph' var '{' GPatt '}'
           | 'graph' iri '{' GPatt '}'
TPatt    ::= Subject Predicate Object
```

```
Subject   ::= var | iri | blankNode
Predicate ::= var | iri
Object    ::= var | iri | blankNode
            | rdfLiteral
Cond      ::=  Expr '=' Expr
            | UnOp Cond
            | Cond BinOp Cond
            | '(' Cond ')'
            | 'isliteral' '(' Expr ')'
            | ...
Expr      ::= var | iri | rdfLiteral
            | 'datatype' '(' Expr ')'
            | ...
UnOp      ::= '!'
BinOp     ::= '&&' | '||'
SubQuery  ::= 'select' Vars
              'where' '{' GPatt '}'
```

Figure: Subset of syntax of SPARQL queries.

Query Containment Problem in SPARQL

- Result of a SPARQL query $\mathtt{Q}$ over an RDF graph $\mathtt{G}$:

  A set of solution mappings in notation:

  $$[\![\mathtt{Q}]\!]_{\mathtt{G}}$$

- $\mathtt{Q}_1$ **is contained in** $\mathtt{Q}_2$ if property $[\![\mathtt{Q}_1]\!]_{\mathtt{G}} \subseteq [\![\mathtt{Q}_2]\!]_{\mathtt{G}}$ holds for **every** RDF dataset $\mathtt{G}$
- Notation:

  $$\mathtt{Q}_1 \sqsubseteq \mathtt{Q}_2$$

- $\mathtt{Q}_1$ is a **sub-query** and $\mathtt{Q}_2$ is a **super-query**

Definition (Query Containment Problem)

For the given two queries $\mathtt{Q}_1$ and $\mathtt{Q}_2$, determine whether $\mathtt{Q}_1$ is contained in $\mathtt{Q}_2$, i.e. whether $\mathtt{Q}_1 \sqsubseteq \mathtt{Q}_2$ holds.

Queries

The query Q_1 is a sub-query of the query Q_2

Q_1 :

```
SELECT ?x ?y ?n
WHERE {
  ?x a :UndergradStudent .
  ?x :takesCourse ?y .
  ?x :name ?n .
  ?x :email ?con .
  ?x :phone ?tel .
}
```

Q_2 :

```
SELECT ?x ?y ?n
WHERE {
  ?x a :UndergradStudent .
  ?x :takesCourse ?y .
  ?x :name ?n .
  ?x :email ?con .
}
```

RDF schema constraints the interpretation of graphs

Query Q_1:

```
1 select ?x where {
2   { ?x :maleHeadOf ?y }
3   union
4   { ?x :femaleHeadOf ?y }
5 }
```

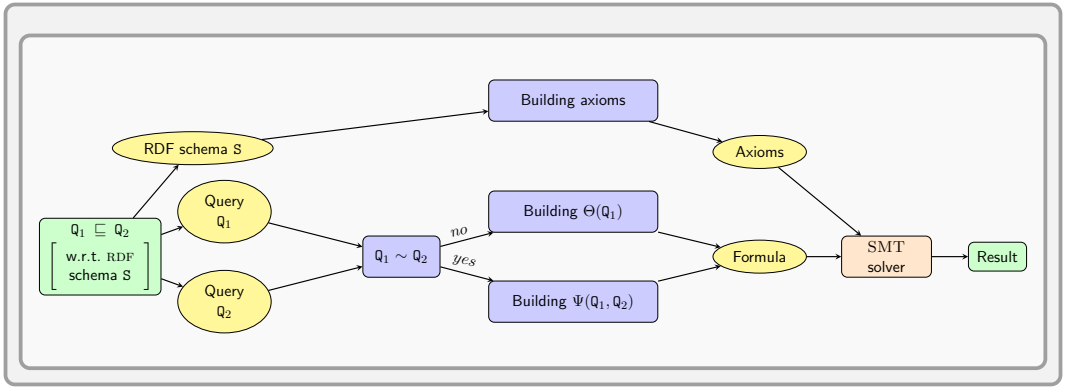
Query Q_2:

```
1 select ?x where {
2   ?x a :Professor
3 }
```

Schema S:

```
1 :maleHeadOf     rdfs:subPropertyOf  :headOf .
2 :femaleHeadOf   rdfs:subPropertyOf  :headOf .
3 :FullProfessor  rdfs:subClassOf     :Professor .
4 :headOf         rdfs:domain         :FullProfessor .
```

Overview of the architecture



Modeling of Graph Patterns

Transforming a SPARQL query into a FOL formula subsumes transforming a graph pattern **recursively** into a formula named Φ:

- SPARQL terms are transformed into the corresponding variables and constants
- For triple pattern `S P O`, uninterpreted predicates β_d and β_n are introduced, indicating the presence of the corresponding triple in RDF graph
- SPARQL graph pattern containing different operators are modeled according to SPARQL semantics, for example, . is modeled with $\wedge$

Modeling must precisely follow SPARQL semantics.
Our modeling was guided by proving soundness (by induction over graph patterns) and completeness (proved by contraposition) of the proposed approach

Relevant variables $\overline{\mathtt{rv}}$. Relation $\sim$. Formula Θ. Formula Ψ.

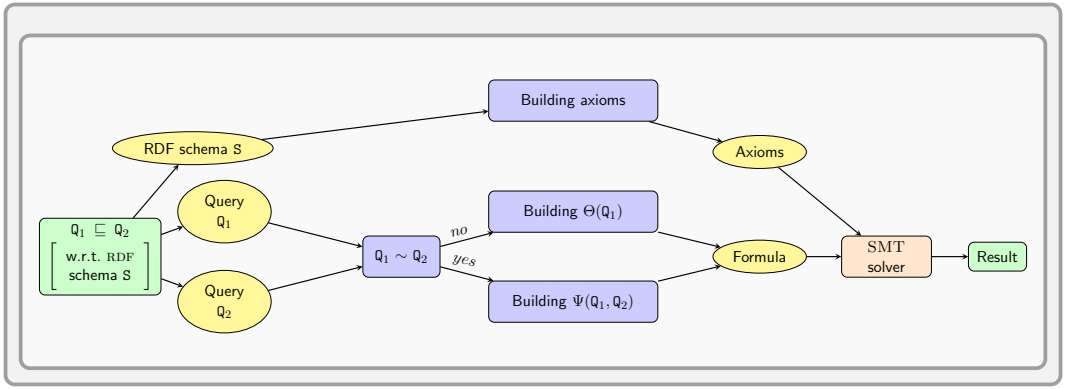
For an RDF dataset $\mathbb{D}$, a query $\mathtt{Q}$ and a mapping μ such that $\mu \in [\![\mathtt{Q}]\!]^{\mathbb{D}}$, all variables from $dom(\mu)$ are called relevant variables and are denoted by $\overline{\mathtt{rv}}$. Queries $\mathtt{Q}_1$ and $\mathtt{Q}_2$ are in relation $\sim$, denoted by $\mathtt{Q}_1 \sim \mathtt{Q}_2$, if relevant variables $\overline{\mathtt{rv}_1}$ and $\overline{\mathtt{rv}_2}$ of these queries are equal, i.e. if $\overline{\mathtt{rv}_1} = \overline{\mathtt{rv}_2}$ holds.
Let $\mathtt{Q}_1$ be a query, let $\overline{rv_1}$ denote variables from $\mathcal{V}$ that correspond to the relevant variables of the query $\mathtt{Q}_1$, and let $\Phi_1(\overline{rv_1})$ correspond to $\mathtt{Q}_1$. Formula Θ is defined as:

$$\neg\big(\exists \overline{rv_1}\ \Phi_1(\overline{rv_1})\big)$$

Let $\mathtt{Q}_1$ and $\mathtt{Q}_2$ be queries, let $\overline{rv_1}$ denote variables from $\mathcal{V}$ that correspond to the relevant variables of the query $\mathtt{Q}_1$, and let $\Phi_1(\overline{rv_1})$ and $\Phi_2(\overline{rv_1})$ correspond to queries $\mathtt{Q}_1$ and $\mathtt{Q}_2$ respectively. Formula Ψ is defined as:

$$\forall \overline{rv_1}\ \big(\Phi_1(\overline{rv_1}) \Rightarrow \Phi_2(\overline{rv_1})\big)$$

Overview of the architecture



Axioms for RDF schema

$$\forall\, s,p,o,i \;\; \beta_n(s,p,o,i) \Rightarrow \beta_d(s,p,o) \qquad \textit{(Default graph)}$$

$$\forall\, s,p,o \;\; \beta_d(s,p,o) \Rightarrow \beta_d(p,\sigma(\mathtt{a}),\sigma(\mathtt{rdf:Property})) \qquad \textit{(rdfD2)}$$

$$\forall\, s,p,o,c \;\; \big(\beta_d(p,\sigma(\mathtt{rdfs:domain}),c) \;\wedge\; \beta_d(s,p,o)\big) \Rightarrow \beta_d(s,\sigma(\mathtt{a}),c) \qquad \textit{(rdfs2)}$$

$$\forall\, s,p,o,c \;\; \big(\beta_d(p,\sigma(\mathtt{rdfs:range}),c) \;\wedge\; \beta_d(s,p,o)\big) \Rightarrow \beta_d(o,\sigma(\mathtt{a}),c) \qquad \textit{(rdfs3)}$$

$$\forall\, s,p,o \;\; \beta_d(s,p,o) \Rightarrow \beta_d(s,\sigma(\mathtt{a}),\sigma(\mathtt{rdfs:Resource})) \qquad \textit{(rdfs4a)}$$

$$\forall\, s,p,o \;\; \beta_d(s,p,o) \Rightarrow \beta_d(o,\sigma(\mathtt{a}),\sigma(\mathtt{rdfs:Resource})) \qquad \textit{(rdfs4b)}$$

$$\forall\, a,b,c \;\; \big(\beta_d(a,\sigma(\mathtt{rdfs:subPropertyOf}),b) \;\wedge\; \beta_d(b,\sigma(\mathtt{rdfs:subPropertyOf}),c)\big) \Rightarrow \beta_d(a,\sigma(\mathtt{rdfs:subPropertyOf}),c) \qquad \textit{(rdfs5)}$$

$$\forall\, p \;\; \beta_d(p,\sigma(\mathtt{a}),\sigma(\mathtt{rdf:Property})) \Rightarrow \beta_d(p,\sigma(\mathtt{rdfs:subPropertyOf}),p) \qquad \textit{(rdfs6)}$$

Axioms for RDF schema

$$\forall\ s, p_1, p_2, o\ \ \big(\beta_d(p_1, \sigma(\texttt{rdfs:subPropertyOf}), p_2)\ \wedge\ \beta_d(s, p_1, o)\big) \Rightarrow \beta_d(s, p_2, o) \qquad (\textit{rdfs7})$$

$$\forall\ c\ \ \beta_d(c, \sigma(\texttt{a}), \sigma(\texttt{rdfs:Class})) \Rightarrow \beta_d(c, \sigma(\texttt{rdfs:subClassOf}), \sigma(\texttt{rdfs:Resource})) \qquad (\textit{rdfs8})$$

$$\forall\ s, c_1, c_2\ \ \big(\beta_d(c_1, \sigma(\texttt{rdfs:subClassOf}), c_2)\ \wedge\ \beta_d(s, \sigma(\texttt{a}), c_1)\big) \Rightarrow \beta_d(s, \sigma(\texttt{a}), c_2) \qquad (\textit{rdfs9})$$

$$\forall\ c\ \ \beta_d(c, \sigma(\texttt{a}), \sigma(\texttt{rdfs:Class})) \Rightarrow \beta_d(c, \sigma(\texttt{rdfs:subClassOf}), c) \qquad (\textit{rdfs10})$$

$$\forall\ a, b, c\ \ \big(\beta_d(a, \sigma(\texttt{rdfs:subClassOf}), b)\ \wedge\ \beta_d(b, \sigma(\texttt{rdfs:subClassOf}), c)\big) \Rightarrow \beta_d(a, \sigma(\texttt{rdfs:subClassOf}), c) \qquad (\textit{rdfs11})$$

$$\forall\ p\ \ \beta_d(p, \sigma(\texttt{a}), \sigma(\texttt{rdfs:ContainerMembershipProperty})) \Rightarrow \beta_d(p, \sigma(\texttt{rdfs:subPropertyOf}), \sigma(\texttt{rdfs:member})) \qquad (\textit{rdfs12})$$

$$\forall\ c\ \ \beta_d(c, \sigma(\texttt{a}), \sigma(\texttt{rdfs:Datatype})) \Rightarrow \beta_d(c, \sigma(\texttt{rdfs:subClassOf}), \sigma(\texttt{rdfs:Literal})) \qquad (\textit{rdfs13})$$

Dealing with Non-Conjunctive Queries: sub-queries, operators `optional` and `union`

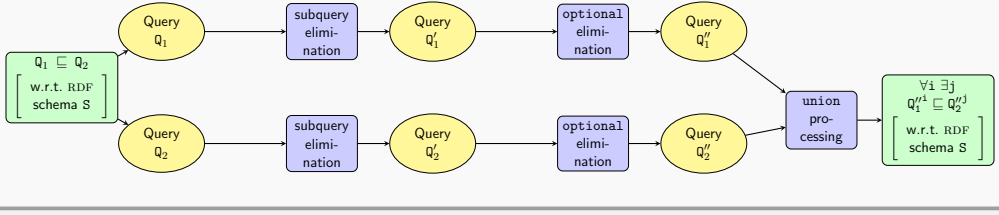


Figure: Reducing the query containment problem to the query containment problems of conjunctive queries with respect to the RDF schema S.

Implementation of **SpeCS**

- Implemented in C++, Lex and Bison used for lexical and syntax analysis
- Query based containment conditions generated in the **SMT-lib** format
- Different external solvers/provers may be used (Z3, Vampire,...)
- **Open source** and available at:
  `http://argo.matf.bg.ac.rs/?content=specs`

Available State-of-the-art Solvers

- Alternation Free two-way μ-calculus (**AFMU**)
  - Reducing the problem to an alternation-free formula of the μ-calculus
- TreeSolver (**TS**)
  - Reducing the problem to the μ-calculus (different encoding)
- SPARQL-Algebra (**SA**)
  - Pattern trees considered as a query execution plan
- Jena-Sparql-Api Graph-Isomorphism based query containment solver (**JSAC**)
  - Bottom-up algebra-tree matching approach
- Query Canonicalisation (**QCan**)
  - Comprehensive procedure for canonicalisation

Table: Comparison of features supported by state-of-the-art solvers.

		SA	AFMU	TS	JSAG	QCan	SpeCS
SPARQL language construct	conj. queries	✓	✓	✓	✓	✓	✓
	`union`		✓	✓	✓	✓	✓
	blank nodes		✓	✓	✓	✓	✓
	projections		✓	✓	✓	✓	✓
	`filter`		✓	✓	✓	✓	✓
	`minus`		✓	✓		✓	✓
	`optional`	✓				✓	✓
	subqueries	✓	✓	✓		✓	✓
	`graph`		✓	✓	✓	✓	✓
	expr. in `select`					✓	✓
	built-in func.		✓	✓	✓	✓	✓
	cycles	✓				✓	✓
QC type	subsumption	✓			✓		✓
	RDF schema		✓	✓			✓
	proj. var. rename				✓	✓	✓

Benchmarks

- Query Containment Benchmark (QC Bench) contains a fixed number of the query containment problems with synthetic queries handcrafted by its designers (76 test cases).
- SQCFramework is a framework that generates customized SPARQL query containment benchmarks based on real SPARQL query logs (78,359 test cases).
- As a side result, SpeCS identified incorrect test cases within both benchmarks, which were manually checked, confirmed and fixed, resulting in better and more reliable benchmarks

Evaluation

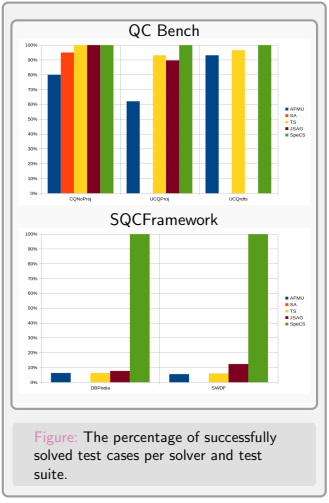


Figure: The percentage of successfully solved test cases per solver and test suite.

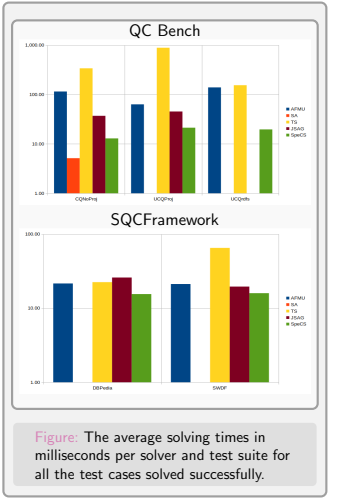


Figure: The average solving times in milliseconds per solver and test suite for all the test cases solved successfully.

Conclusions and further work

- Detailed results and discussion is available in the paper
- The evaluation shows that SpeCS is highly efficient and that compared to the state-of-the-art solvers, it gives more precise results in a shorter amount of time. In addition, SpeCS has the highest coverage of the supported language constructs.
- We have manually proved soundness and completeness of the proposed approach (for conjunctive queries) and are planning to do that formally (within a proof assistant)

Refactoring of Embedded SQL — Motivation

- There is a number of approaches for dealing with equivalence of either pairs of imperative code fragments or pairs of SQL statements
- Database driven applications: simultaneous changes (changes that include both SQL and a host language code)
- Such changes can preserve the overall equivalence without preserving equivalence of these two parts considered separately
- We propose an automated approach for dealing with equivalence of programs after such changes

Example refactoring

```
int pending_deposit(int account_id){
  account_sqc = account_id;
  EXEC SQL SELECT T1.a_avail_blnc
                  -T1.a_pending_blnc
   INTO :pending_deposit_sqc

   FROM account AS T1
   WHERE T1.a_account_id=:account_sqc
  if (sqlca.sqlcode) goto errorexit;
  return pending_deposit_sqc;

}
```

```
int pending_deposit(int account_id){
  account_sqc = account_id;
  EXEC SQL SELECT T1.a_pending_blnc,
                  T1.a_avail_blnc
   INTO :pending_blnc_sqc,
        :avail_blnc_sqc
   FROM account AS T1
   WHERE T1.a_account_id=:account_sqc;
  if (sqlca.sqlcode) goto errorexit;
  return avail_blnc_sqc
         -pending_blnc_sqc;
}
```

Figure: Calculation moved form SQL to the hoste language.

- We propose axiom-based semantics for SQL queries that can be linked with semantics of imperative programs and used for automated reasoning about functional equivalence of SQL based functions
- Aims at proving equivalence in case when neither the SQL code nor the host language code are equivalent individually, but their combination is
- The approach generates equivalence conditions that can be efficiently checked using SMT solvers or first-order logic provers

From code to FOL formula

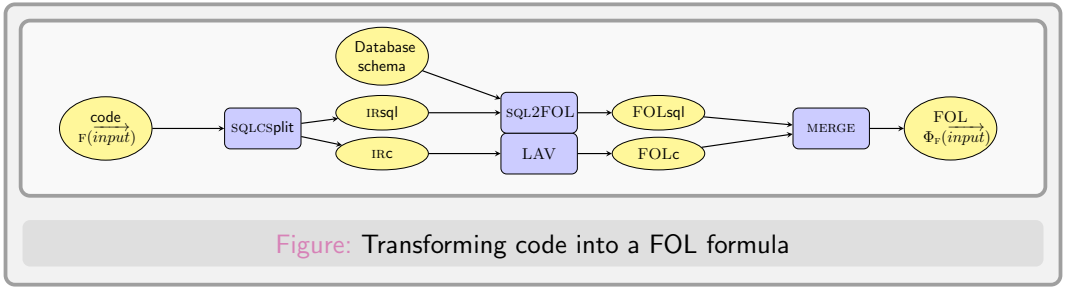


Figure: Transforming code into a FOL formula

SQLav architecture

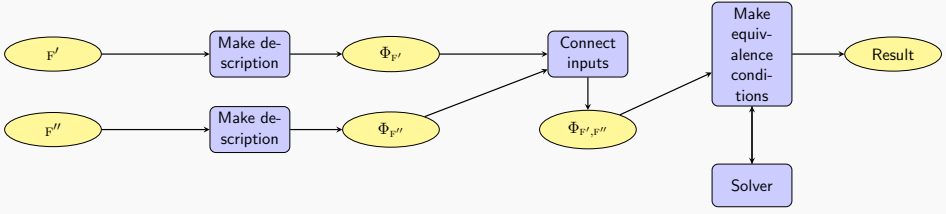


Figure: Architecture overview.

Implementation

- We implemented the proposed approach as a framework SQLav, which is publicly available and open source http://argo.matf.bg.ac.rs/?content=lav
- The implementation follows the described architecture, and uses tool lex and bison for lexical and syntax analysis
- It uses the tool **LAV**, which was upgraded for this purpose
- Formulas are generated in **SMT-lib** format
- It can use different proves/solvers (Z3, Vampire,...)
- We also implemented GitHub Actions

Evaluation

- No standard benchmarks available
- No other tools dealing with this problem
- We made a benchmark consisting of 280 test cases for evaluating our tool
- Benchmark contains different SQL constructs and different interactions with a host language
- Evaluation assesses efficiency and preciseness of the proposed approach
- Evaluation shows that the proposed approach can be efficiently used in many different contexts
- The approach gives useful hints in cases when equivalence is not proved

Logical status of the modeling

- The presented approach is not complete, but still it contains relevant information for proving a number of important cases of equivalence
- Our confidence in soundness relies on relatively simple and intuitively acceptable axioms schemes used and also on the fact that not all our goals are proved by refutation, but some goals are rejected — we have to work on this further
- In several aspects the considered problem is naturally expressed in terms of higher order logic rather then in terms of FOL — we made an effort to perform all automated reasoning within FOL in order to use efficient available FOL provers and SMT solvers

Conclusions

- The tool SpeCS has excellent results compared to other state of the art QC solvers
- Further work: proof of soundness and completeness within a proof assistant
- Further work: consider SPARQL within a host language
- We are the first to tackle the important problem of connecting semantics of SQL with semantics of imperative host language, so we also built a big benchmark as a starting point for further comparisons and evaluations
- Adding support for automated generation of test cases from the model given by an SMT solver