DATABASE THEORY

Lecture 20: Outlook

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 5th July 2022

More recent versions of this slide deck might be available.
For the most current version of this course, see
`https://iccl.inf.tu-dresden.de/web/Database_Theory/en`

Database Theory in Practice?

We have seen many query languages:

- CQ, FO, (2)RPQ, C(2)RPQ, Datalog, linear Datalog, semipositive Datalog, . . .

. . . and many optimisation techniques:

- optimisation of tree-like queries
- CQ containment and equivalence
- Datalog implementation techniques

Is any of this relevant in practice?

Review: FO, relational algebra, and SQL

The following are essentially equivalent:

- First-order queries
- Relational algebra queries
- "Basic" SQL queries

where different applications may use slightly different variants
(named vs. unnamed perspective; tuple-relational calculus; domain independent vs. active domain semantics; . . .)

We get CQs when restricting to SELECT-PROJECT-JOIN queries.

$\leadsto$ All RDBMSs implement FO queries, and CQs as special case

Recursive Queries in SQL

The SQL'99 standard supports recursive queries through the `WITH RECURSIVE` construct.

- IDB predicates are called common table expressions (CTE) in SQL
- A CTE is defined by a single SQL query, which can use the CTE recursively
- The standard defines a fixed point semantics, similar to Datalog
- Widely supported today (IBM DB2, PostgreSQL, Oracle 11g R2, MS SQL Server, . . .), but implementations vary and don't conform to a common standard so far

Recursive Queries in SQL: Example

Find all ancestors of Alice:

```
WITH RECURSIVE ancestor(young, old) AS (
        SELECT parent.young, parent.old FROM parent
      UNION ALL
        SELECT ancestor.young, parent.old
        FROM ancestor, parent
        WHERE ancestor.old = parent.young
)
SELECT * FROM ancestor WHERE ancestor.young = 'alice';
```

Notes:

- `UNION ALL` keeps duplicates, which leads to a multiset (bag) semantics that may cause termination problems.
- Many RDBMSs will fail to push the selection `ancestor.young = 'alice'` into the recursion; modifying the CTE definition to start from `'alice'` would help them.

Expressive Power of Recursive SQL

The expressive power of recursive SQL is not easy to determine:

- A CTE uses only a single IDB predicate, but it can use unions
- `UNION ALL` enforces a multiset semantics
- SQL subsumes FO queries (including negation!)
- SQL has other features, e.g., adding numbers
- Specific RDBMSs have own extensions or restrictions

Expressive Power of Recursive SQL

The expressive power of recursive SQL is not easy to determine:

- A CTE uses only a single IDB predicate, but it can use unions
- `UNION ALL` enforces a multiset semantics
- SQL subsumes FO queries (including negation!)
- SQL has other features, e.g., adding numbers
- Specific RDBMSs have own extensions or restrictions

Some relevant questions:

- Can I use negation to filter duplicates during recursion?

  SQL allows this, but implementations like MS SQL Server return wrong results when trying this (unsuitable implementation approach that operates "depth-first" tuple-by-tuple using separate "stacks").
- Can I use the CTE more than once in a recursive term?

  SQL allows this, but not all RDBMSs support it. Even RDBMSs that allow it may not always implement it correctly, so some care is needed.

Expressive Power of Recursive SQL (2)

SQL is too powerful for a declarative recursive query language:

- Combination of negation and recursion is hard to define and implement.
- Functions such as addition can extend the active domain.

$\leadsto$ non-declarative approach to recursion (Turing complete)

$\leadsto$ all implementations allow non-terminating queries

With care, one can still formulate sane queries.

Expressive power in terms of Datalog:

- Minimal: linear Datalog with bounded recursion depth (can still be useful, e.g., for navigating hierarchies)
- Maximal: arbitrary semi-positive Datalog with successor order, and beyond

Recursion in SQL: Conclusions

Mixed picture of recursion in SQL:

- SQL'99 supports arbitrary Datalog
- Practical implementations are ad hoc and rather limited
- No simple & terminating queries with unbounded recursion
- Some implementations seem to support at least linear Datalog in a clean way
  (e.g., PostgreSQL supports `UNION` and duplicate elimination in recursive CTEs, using a special case of semi-naive evaluation)
- Online documentation mostly fails to clarify restrictions

Recursion in SQL: Conclusions

Mixed picture of recursion in SQL:

- SQL'99 supports arbitrary Datalog
- Practical implementations are ad hoc and rather limited
- No simple & terminating queries with unbounded recursion
- Some implementations seem to support at least linear Datalog in a clean way (e.g., PostgreSQL supports `UNION` and duplicate elimination in recursive CTEs, using a special case of semi-naive evaluation)
- Online documentation mostly fails to clarify restrictions

Recursive CTEs are not the only option:

- Oracle has a proprietary SQL extension `CONNECT BY`
- similar to Transitive Closure operator in FO queries
- designed for linear recursion

Oracle speaks of "subquery factoring" when using CTEs.

Practical Recursion Beyond SQL

SQL support for recursion is a bit shaky
⇝ how about other types of DBMSs?

Recursion plays a role in a number of distinct areas, including:

- Datalog implementations
- XQuery and XPath query languages for XML
- SPARQL query language for RDF
- Graph query languages

Datalog Implementation in Practice

Dedicated Datalog engines as of 2022 (incomplete):

- VLog Fast in-memory Datalog materialisation with bindings to several databases, including RDF and RDBMS (free, co-developed at TU Dresden)
- Graal In-memory rule engine with RDBMS bindings (free)
- Gringo Fast Datalog-based grounder for answer set programming (free)
- RDFox Fast in-memory RDF database with runtime materialisation and updates (commercial)
- Vadalog Closed-source engine with several extensions (commercial)
- Llunatic PostgreSQL-based implementation of a rule engine (free, discontinued)
- SociaLite and EmptyHeaded Datalog-based languages and engines for social network analysis
- DeepDive Data analysis platform with support for Datalog-based language “DDlog”
- DLV Answer set programming engine that is usable on Datalog programs (commercial)
- Datomic Distributed, versioned database using Datalog as main query language (commercial)
- LogicBlox Big data analytics platform that uses Datalog rules (commercial, discontinued)
- E Fast theorem prover for first-order logic with equality; can be used on Datalog as well
- . . .

$\leadsto$ Extremely diverse tools for very different requirements

Querying RDF Graphs with SPARQL

SPARQL Protocol and RDF Query Language

- Query language for RDF graphs (roughly: labelled, directed graphs)
- W3C standard, currently in version 1.1 (2013)
- Widely used for accessing RDF databases

Querying RDF Graphs with SPARQL

SPARQL Protocol and RDF Query Language

- Query language for RDF graphs (roughly: labelled, directed graphs)
- W3C standard, currently in version 1.1 (2013)
- Widely used for accessing RDF databases

Structure of a simple SPARQL query:

```
SELECT <variable list> WHERE { <pattern> }
```

- `<pattern>` is a basic graph pattern: a list of “triples” of the form “`subject predicate object .`” (denoting an edge from `subject` to `object` labelled by `predicate`)
- Patterns may contain variables (marked by prefix `?`) that can be selected
- Many other features (more complex conditions in queries, limit & offset, grouping & aggregation, . . .)

SPARQL Query Example

Find people whose parents were born in the same city in Saxony, and return them together with that city:

```
PREFIX ex: <http://example.org/>
SELECT ?person ?city
WHERE {
    ?person ex:hasMother ?mother .
    ?person ex:hasFather ?father .
    ?mother ex:bornIn ?city .
    ?father ex:bornIn ?city .
    ?city ex:locatedIn ex:Saxony .
}
```

Essentially a conjunctive query with ternary EDB predicates written in a simple text-based syntax

SPARQL and Recursion

Since version 1.1, SPARQL supports C2RPQs:
Property Path Expressions

Regular expression syntax:

- Single letter: name (URI) of a property (predicate) in RDF
- Converse ℓ^- of letter ℓ is written as $\hat{}\ell$
- Sequence ($\circ$) is /, alternative ($+$) is |, zero-or-more is *
- Other features: optional ?, one-or-more +, atomic negation !

Example:

```
PREFIX ex: <http://example.org/>
SELECT ?person ?ancestor
WHERE {
  ?person ( (ex:hasMother|ex:hasFather)+ ) ?ancestor .
}
```

Recursion in SPARQL: Conclusions

Widely supported feature of most modern RDF databases

- Set-based semantics that agrees with C2RPQs
- Typically implemented in a declarative way (no operational extensions)
- Guaranteed to terminate, given sufficient resources
- Performance depends on implementation and data (not all implementations have a good optimiser for property paths)
- Example systems: BlazeGraph, OpenLink Virtuoso, Stardog, Amazone Neptune, . . .
- Frequently used (in particular on Wikidata, where around 20% of SPARQL queries used * in Jan–Mar 2018 [Malyshev et al., ISWC 2018])

Recursion in other Graph Databases

Graph databases support recursive queries, but there is no standard query language
$\leadsto$ sometimes not fully clear what is supported/moving target

Example: Cypher query language in Neo4J

```
MATCH (p)-[r:HasMother|HasFather*]->(a)
WHERE p.name='Alice'
RETURN p,r,a
```

- Support for retrieving matched paths (`r` in example)
- Additional graph search features (shortest path, limited recursion, etc.)
- No full support for RPQs, since stars cannot be applied to complex expressions
- Query matching is based on isomorphism rather than homomorphism

  (does not make a difference when checking the existence of simple paths, but does make a difference for CQs and for counting queries)

For further information on graph databases and their features, see the course “Knowledge Graphs” at TU Dresden (offered in winter term).

Recursion in XML Document Processing

XML a W3C standard for a document markup language

- XML is used for markup and data representation
- XML documents can be interpreted under a tree-shaped Document Object Model (DOM)
- DOM tree is an ordered tree where each node has a type, and optionally also attribute values

The XML query language XPath defines ways to query XML DOMs

- W3C standard now in version 3.0 (2014); many practical implementations based on XPath 1.0
- Key concept: expressions to select (query) nodes and attributes in a DOM tree
- Recursion is important for navigating trees

XPath Expression Examples

XPath expressions navigate the DOM tree by using natural binary relations among nodes, called axes, such as “child” and “descendant.”

Example XPath expressions:

- `/A/B` nodes of type B that are children of a node of type A that is the root of the DOM tree
- `A//C` arbitrary descendants of the a node of type A that is the start node (context node) for the query
- `//C[./D/E]/F` nodes of type F that are the child of a node of type C anywhere in the DOM, where the C-node has a D child that has an E child.

There are many further features related to attribute selection and use of other axes

XPath: Expressive Power

XPath is related to 2RPQs

- There are some differences between DOM trees and words
- Many XPath location steps could be written in 2RPQ

Predicates in square brackets are used to test additional path-like conditions for a node

- Example: `A[.//B]` only matches A-type nodes that have a descendant of type B
- Corresponds to unary sub-2RPQs of the form $\exists y.E(x, y)$ that test if a node x has an E-path to some other node

$\leadsto$ not expressible in (C)2RPQs without further extensions

Recursion in XPath: Conclusions

XPath: XML navigation base on path queries

- Declarative, set-based semantics
- Standardised in several versions
- Many implementations (program libraries, some DBMS)
- Large number of features – hard to analyse theoretically

Related approaches:

- XQuery: extension of XPath with computational features
- CSS Selectors: simple query language for navigating HTML documents

Summary and Outlook

Summary: Queries

We have covered three main topics:

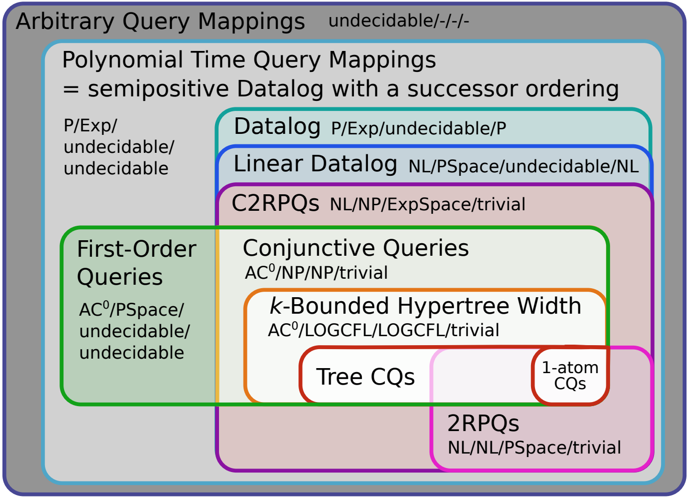
- first-order queries
- Datalog
- path query languages

looking at the following main aspects:

- expressive power
- complexity of query answering
- complexity/feasibility of perfect query optimisation
- some algorithmic approaches

Equal focus on results and methods
$\leadsto$ understanding why something holds

The Ultimate Big Picture



Legend: Data compl./Comb. & Query compl./Equivalence & containment/Emptiness

The Big Picture: Notes for Offline Reading

- Given complexities usually are upper and lower bounds (“complete”), though AC^0 is just an upper bound
- “Linear Datalog” refers to the strict definition given in the course. Some authors consider a final CQ “on top” of linear Datalog programs, but this does not change anything (see below).
- The “-” for arbitrary query mappings mean that these problems are not defined (we have no query expressions that could be the input of an algorithm, just mappings).
- Some complexities given were not shown, including P-completeness of Datalog emptiness (left as exercise).
- Most complexities for semipositive Datalog with a successor ordering are easily obtained from Datalog using the fact that the required negated EDB predicates and ordering facts can be added to a given database in polynomial time.

The Big Picture: Notes for Offline Reading

Emptiness of semipositive Datalog with a successor ordering is not quite so obvious …

Proof sketch:

- Emptiness of the intersection of two context-free grammars G_1 and G_2 is undecidable.
- The word problem of context-free grammars is in P.
- A database can encode a word if it is a linear chain using binary letter predicates. This can be checked in P.
- Semipositive Datalog with successor captures P, so there is a Boolean query P_{G_1,G_2} in this language that decides if the database encodes a word that is in G_1 and G_2.
- The emptiness problem of P_{G_1,G_2} is equivalent to the emptiness problem for $G_1 \cap G_2$.

The Big Picture: Notes for Offline Reading

The fact that linear Datalog extends C2RPQ is not obvious either:
how can we express conjunctions over IDBs there?

Proof sketch:[1]

- The C2RPQ can be viewed as a CQ over IDBs that are defined by linear Datalog programs obtained for 2RPQs
- Without loss of generality, we assume that each of these linear Datalog programs uses differently named IDB predicates
- We transform this CQ over IDB atoms step by step
- In each step, process two IDB atoms $\mathsf{Q}(x_1, \ldots, x_n)$ and $\mathsf{R}(y_1, \ldots, y_m)$
  - Replace them by a single new atom $\mathsf{R}'(x_1, \ldots, x_n, y_1, \ldots, y_m)$
  - Use linear rules that consist of all rules used for defining Q together with modified versions of the rules for R that “remember” a binding for Q while deriving facts about R.
- Continue until only one IDB is left in the conjunction.

---

[1] For details on a similar proof, see Theorem 3 in P. Bourhis, M. Krötzsch, S. Rudolph: Reasonable Highly Expressive Query Languages, Proc. IJCAI 2015.

Summary: Dependencies

Dependencies

- provide useful information about the database schema
- can be used for defining (recursive) views and integrating data
- generalise many concrete types of DB dependencies

The chase provides a principled bottom-up method for computing universal models and answering queries.

Query entailment under dependencies is undecidable,
but we have seen three approaches to overcome this:

- Finiteness: universal models are finite (several acyclicity notions)
- Bounded treewidth: universal models have bounded treewidth (several guardedness conditions)
- First-order rewritability: queries can be finitely rewritten (linearity and other conditions)

Conclusions

The relational data model remains the most widely used general data model, but alternative data models are now also relevant:

- "noSQL" data models (graphs, trees, documents, map, . . .)
- All major RDBMS vendors have products in this space, sometimes based on their RDBMSs, sometimes not
- Revival of specialised stores and data models

The same basic theory applies to relational and non-relational DBMSs:

- all data models can be viewed as relational
- fundamental query types re-appear in many settings (CQs, path queries, . . .)
- non-relational DBMS are taking the lead in realising more advanced concepts (recursive queries, clean set-based semantics)

What's next?

Current data management landscape is extremely dynamic and hard to predict – interesting times!

- Many further topics not covered here (data stream processing, distributed models of computation, analytical queries, . . .)
- Many theoretical questions remain open (further query languages, constraints/ontologies, algorithms, . . .)

A wider view is key to success:

- Practitioners need to know their tools and be ready to combine them into custom solutions
- Theoreticians need to combine methods from distinct areas and re-integrate practical developments

What's next?

Current data management landscape is extremely dynamic and hard to predict – interesting times!

- Many further topics not covered here (data stream processing, distributed models of computation, analytical queries, . . .)
- Many theoretical questions remain open (further query languages, constraints/ontologies, algorithms, . . .)

A wider view is key to success:

- Practitioners need to know their tools and be ready to combine them into custom solutions
- Theoreticians need to combine methods from distinct areas and re-integrate practical developments

Basic principles are more important than short-lived technology trends,
but practice and theory must interact to create relevant and meaningful solutions.