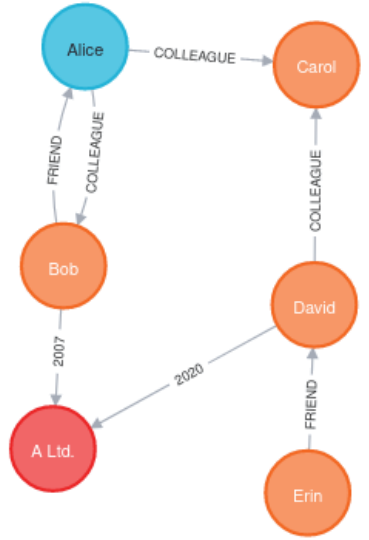
Lambers et al.: Model-Based Testing of Read Only Graph Queries (ICSTW'20)

Márton Elekes

Budapest University of Technology and Economics
Department of Measurement and Information Systems
ftsrg Research Group

Short intro to graph databases



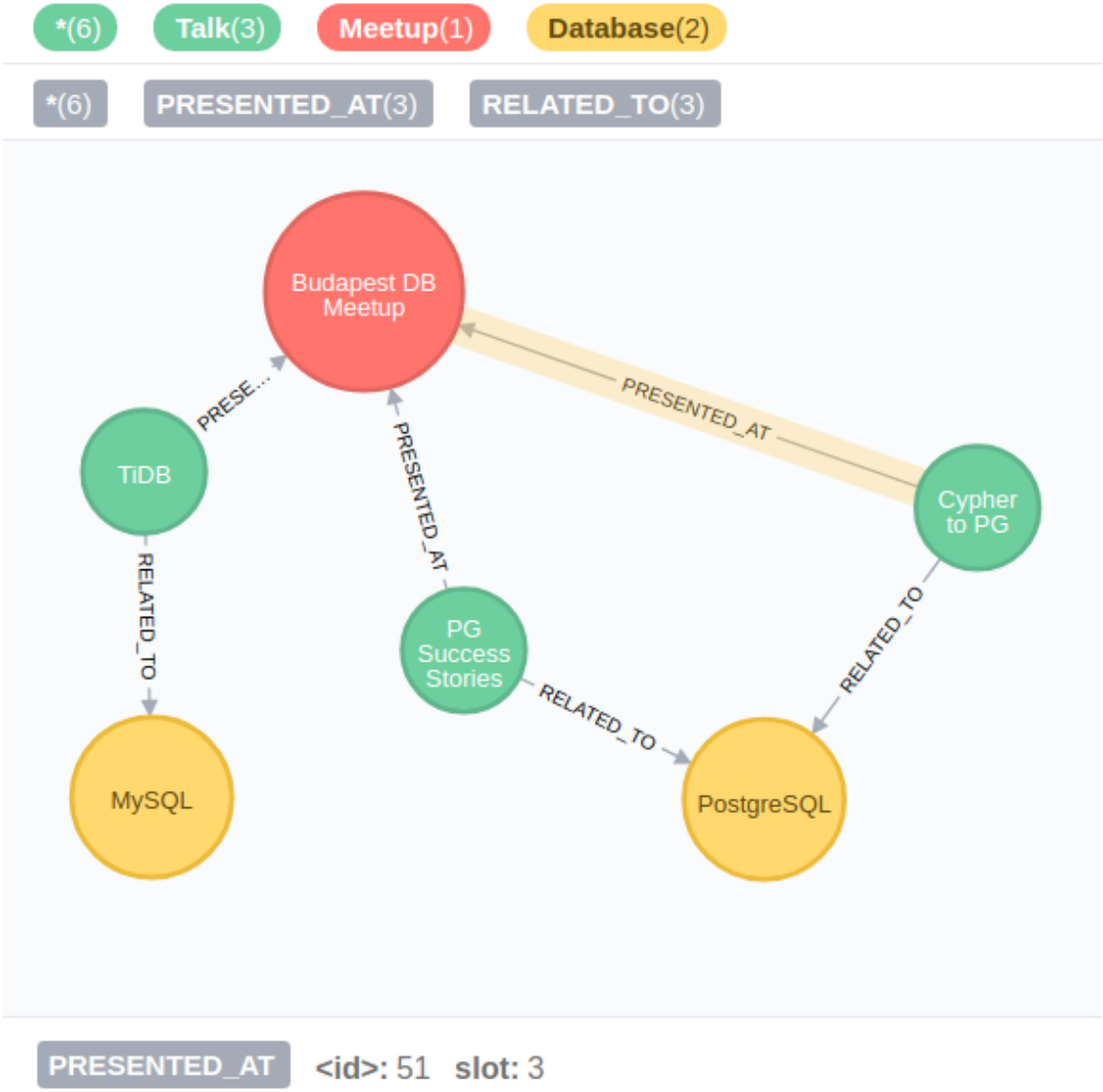
Property graph databases

NoSQL family

Data model:

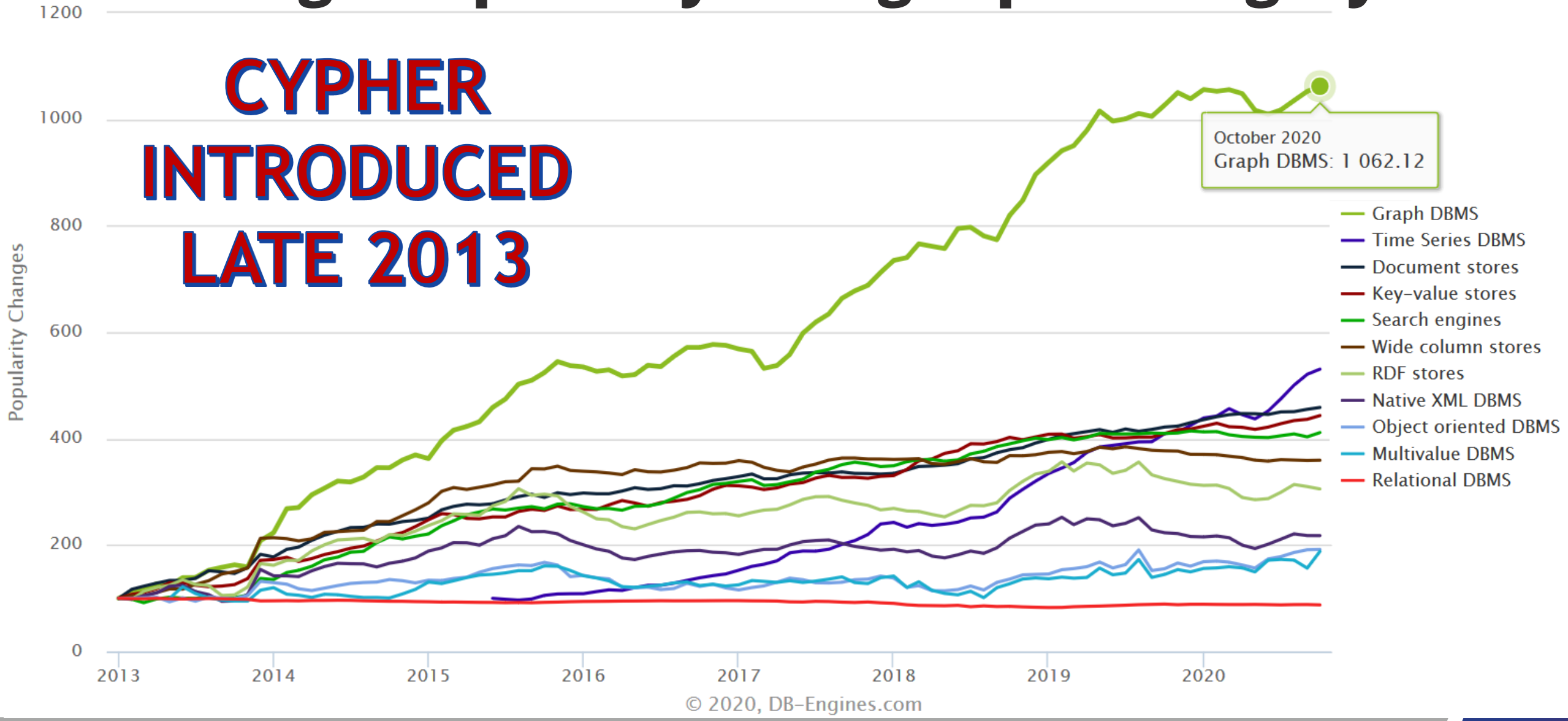
- nodes
- edges
- properties

#1 query approach:
graph pattern matching



(source)

Rankings: Popularity changes per category



Cypher → openCypher, … → GQL (ISO/IEC)

Cypher: query language of the Neo4j graph database.

„Cypher is a declarative, SQL-inspired language for describing patterns in graphs visually using an ascii-art syntax.”

```
MATCH (testing:Topic)<-[:RELATED_TO]-(:Paper)
                      -[:PRESENTED_AT]->(l:Lecture)
WHERE l.date = '2 December 2020'
RETURN testing
```

„The  openCypher project aims to deliver a full and open specification of the industry’s most widely adopted graph database query language: Cypher.” (2015)

GQL Graph Query Language:
ISO/IEC standard proposal for a graph query language (June 2019)

(source)

openCypher systems

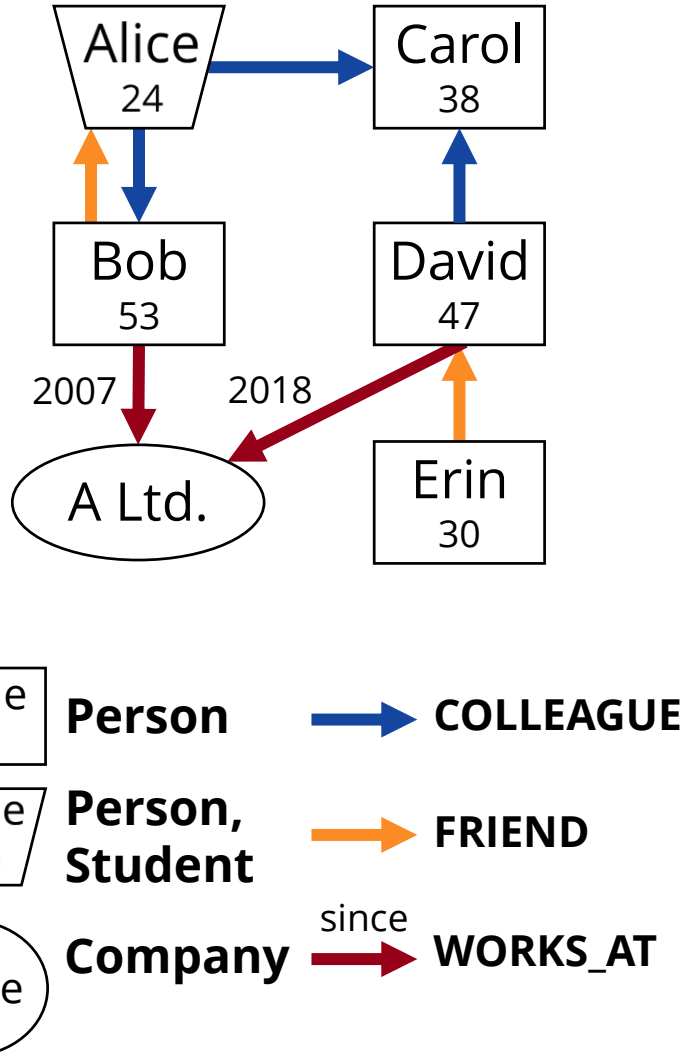
- Increasing adoption
- Relational databases:
  - SAP HANA
  - AGENS Graph
- Research prototypes:
  - Graphflow (Univesity of Waterloo, Canada)
  - ingraph (incremental graph engine @ BME)

(Source: Keynote talk @ GraphConnect NYC 2017)

(source)

Property graphs

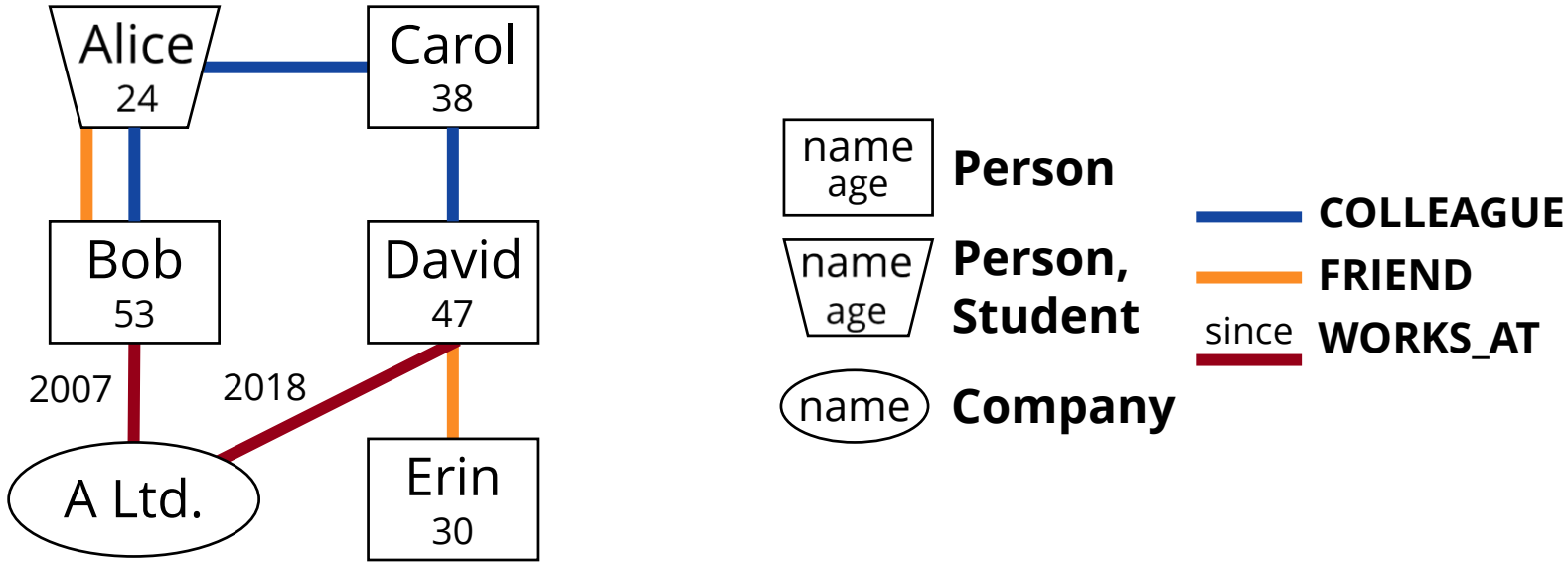
- Textbook graph: $G = (V, E)$
  - Dijkstra, Ford-Fulkerson, etc.
  - Homogeneous nodes
  - Homogeneous edges
- Extensions
  - Labelled nodes
  - Typed edges
  - Properties
- The schema is implicit
- Very intuitive
  - *Things* and *connections*



Graph queries in Cypher

- Subgraph matching and graph traversals
- Example: Alice's colleagues and their colleagues

```
MATCH (p1:Person {name: 'Alice'})-[c:COLLEAGUE*1..2]-(p2:Person)
RETURN p2.name
```



Applications of graph-based modelling

- Social networks
- Financial fraud detection
- Serving personalized recommendations
- Systems modeling: e.g. UML
- Pharmaceutics

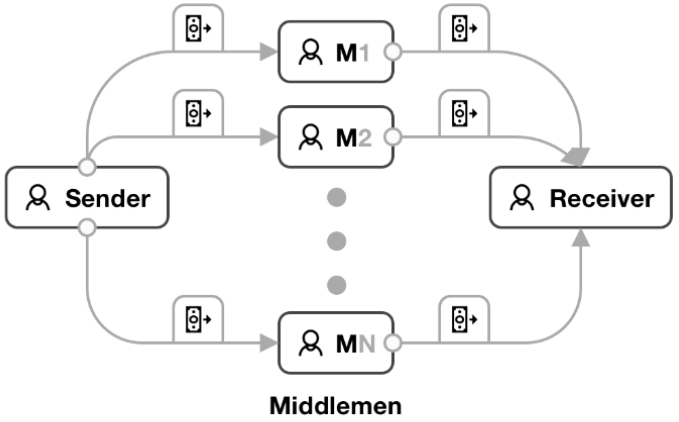
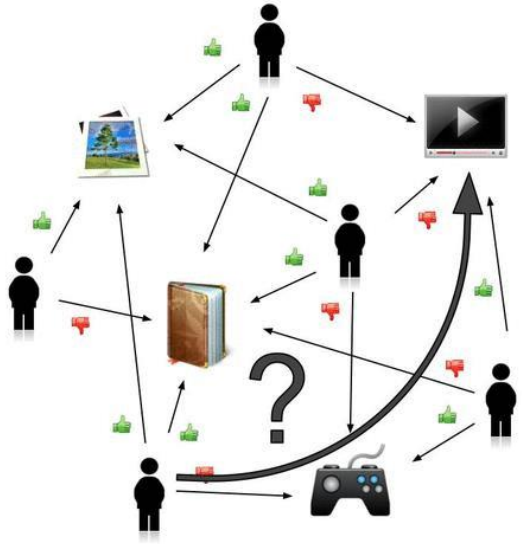
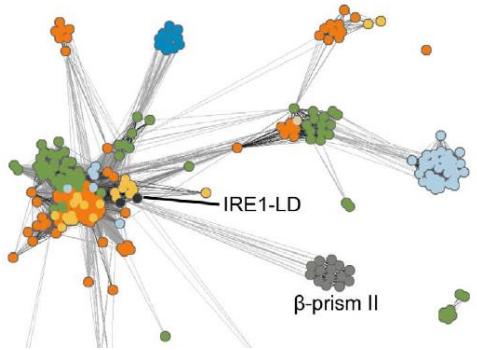


Figure 3.1: Smurfing fraud scenario





Sources: Karimov Karim: Graph-based fraud detection on financial transactions
An example of predicting of the user's rating using collaborative filtering, https://commons.wikimedia.org/wiki/File:Collaborative_filtering.gif
β-Propeller Blades as Ancestral Peptides in Protein Evolution, https://www.researchgate.net/figure/Cluster-map-of-b-propellers-and-their-potential-homologs-Dots-represent-proteins_fig9_258062452

Model-Based Testing of Read Only Graph Queries

2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)

Model-Based Testing of
Read Only Graph Queries

Leen Lambers
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
Leen.Lambers@hpi.de

Sven Schneider
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
Sven.Schneider@hpi.de

Marcel Weisgut
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
Marcel.Weisgut@student.hpi.de

***Abstract*—Modern software applications often interface with graph-structured data managed by graph databases queried with specific graph query languages. Writing, understanding, and maintaining graph queries can be difficult and error-prone. Important decisions taken by the software application are in many cases based on the outcome of these graph queries. If test design does not account for the complexity incorporated in these queries, considerable parts of the business logic of such applications will not be tested appropriately.**

We aim at tackling the challenge of developing dedicated testing techniques for graph queries. In particular, we present a first model-based testing approach for read only queries supporting automated test creation, execution, and evaluation. We model queries using a well-established graph logic, being expressively equivalent to first-order logic on graphs. We develop a first coverage criterion requiring the presence of a test case returning an empty query result as well as the presence of a test case returning a non-empty result. We present the architecture of our model-based testing approach, which is supported by a first implementation. We illustrate our approach with complex read only graph queries from an industrial benchmark case study.

***Index Terms*—automatic testing, Logic testing, Software testing, NoSQL databases**

I. Introduction

Important parts of the business logic in *database applications* are implemented in database interactions. It is therefore indispensable to take these interactions explicitly into account when creating tests checking for functional correctness [1], [2]. Within the regular control flow of the application, read only queries (not changing the graph) need to read correct data from the database and, in addition, write queries need to correctly modify read data. Also, database developers need to have tests available for checking different query evaluation strategies [3]. For the *relational type of queries*, dedicated automated testing techniques exist, aiming at checking in this sense functional correctness of queries (cf. section II).

Modern software applications often rely on graph query languages to interface with their data managed by graph databases [4], [5]. These databases focus the manipulation of data in the form of graphs. They are therefore often used if relationships between entities play an equivalently or even more important role than the entities themselves. For *graph query languages* used to interface with these graph databases dedicated automated testing techniques do not exist yet, although data quality [6] is very important in particular in the application areas of graph databases such as social networks [7], biological networks [8], and software analysis [9]. We aim at developing dedicated automated testing techniques for checking functional correctness of graph queries.

We present in this paper a first step towards this goal by developing a *model-based testing* [10] framework for *read only graph queries*. We thereby aim at *finding functional faults in the queries* themselves as well as in the evaluation mechanism *of the graph database*. As opposed to the relational case with SQL as a standardized query language, there does not exist an established standard query language for graph databases yet. Our overall aim therefore is to develop a test approach for different graph databases with varying query languages, which is *one reason* for following a model-based testing approach. In this paper, we exemplarily focus the graph database Neo4J [11] and its query language Cypher to demonstrate our approach. In the long term, we will then be able to reuse parts of the test generation infrastructure and in particular compare different query implementations as well as evaluation mechanisms of different graph databases. *Another advantage* of following a model-based approach is that we can abstract from testing specific implementation details such as aggregation operators when our aim is to focus on graph-structural issues.

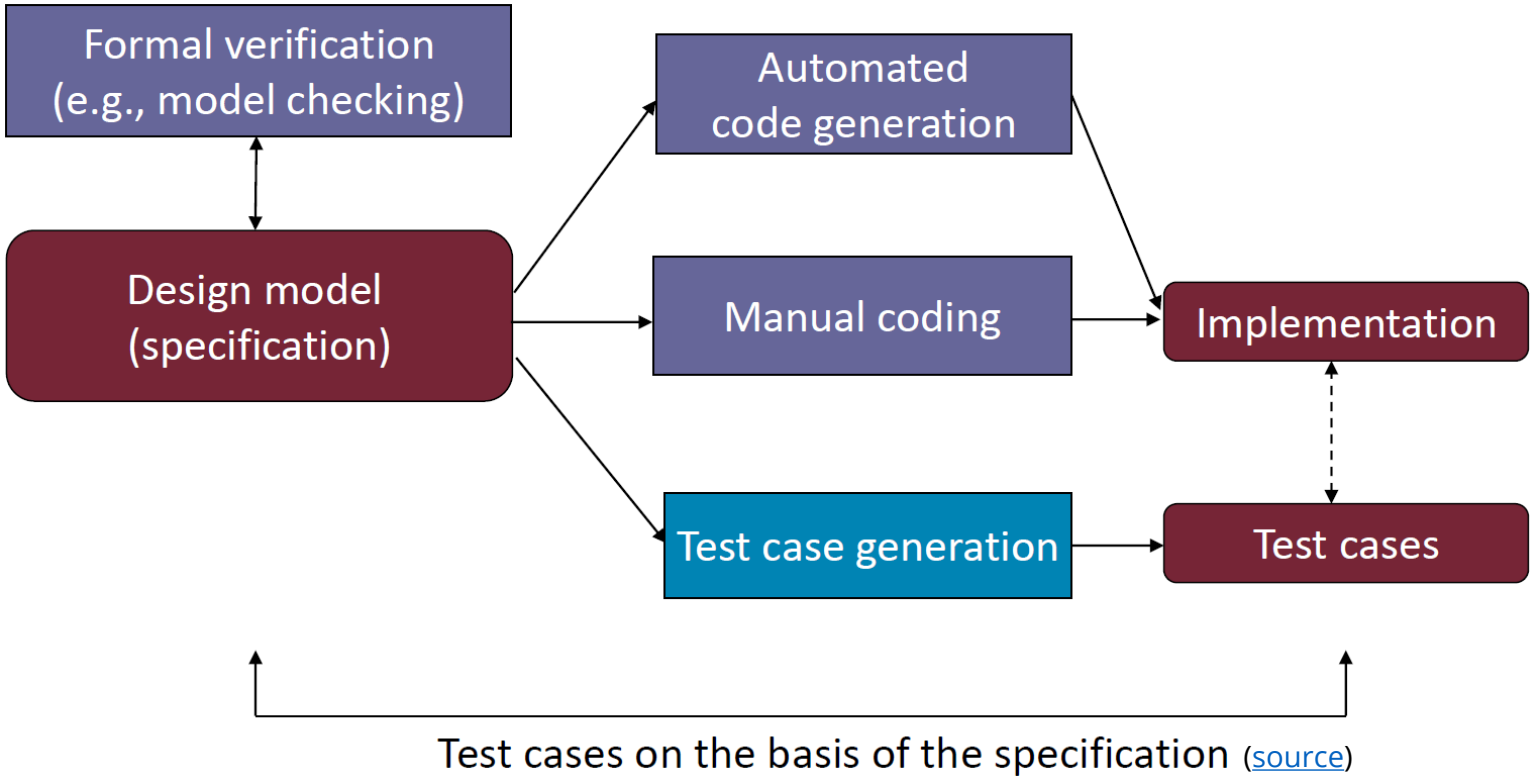
In our model-based testing approach as illustrated in Figure 1, the tester describes complex read only graph queries [12] based on some natural language specification of the query using a *dedicated graph logic GL* [13], [14]. It was shown already in previous (in part own) work [15]–[18] that the graph logic GL (being equivalent to first-order logic on graphs as shown in [13]) is very appropriate w.r.t. modeling complex read only graph queries. In addition the tester can select a first *coverage criterion* that we developed for queries expressed within this graph logic requiring the presence of a test case returning an empty query result as well as the presence of a test case returning a non-empty result. Then *absence sen cases* satisfying this criterion are automatically generated. From them our *adapter component* performs automatic concretion of abstract test inputs (being graphs) to concrete test inputs (being actual graph databases) as well as abstraction of concrete query results obtained for the *implemented queries* for automatic comparison with the expected abstract result.

We illustrate and present a first evaluation of our approach using complex read only graph queries (cf. Figure 2 for

978-1-7281-1075-2/20/$31.00©2020 IEEE
DOI 10.1109/ICSTW50294.2020.00022

Model-Based Testing of Read Only Graph Queries

- Model-based testing: test case generation

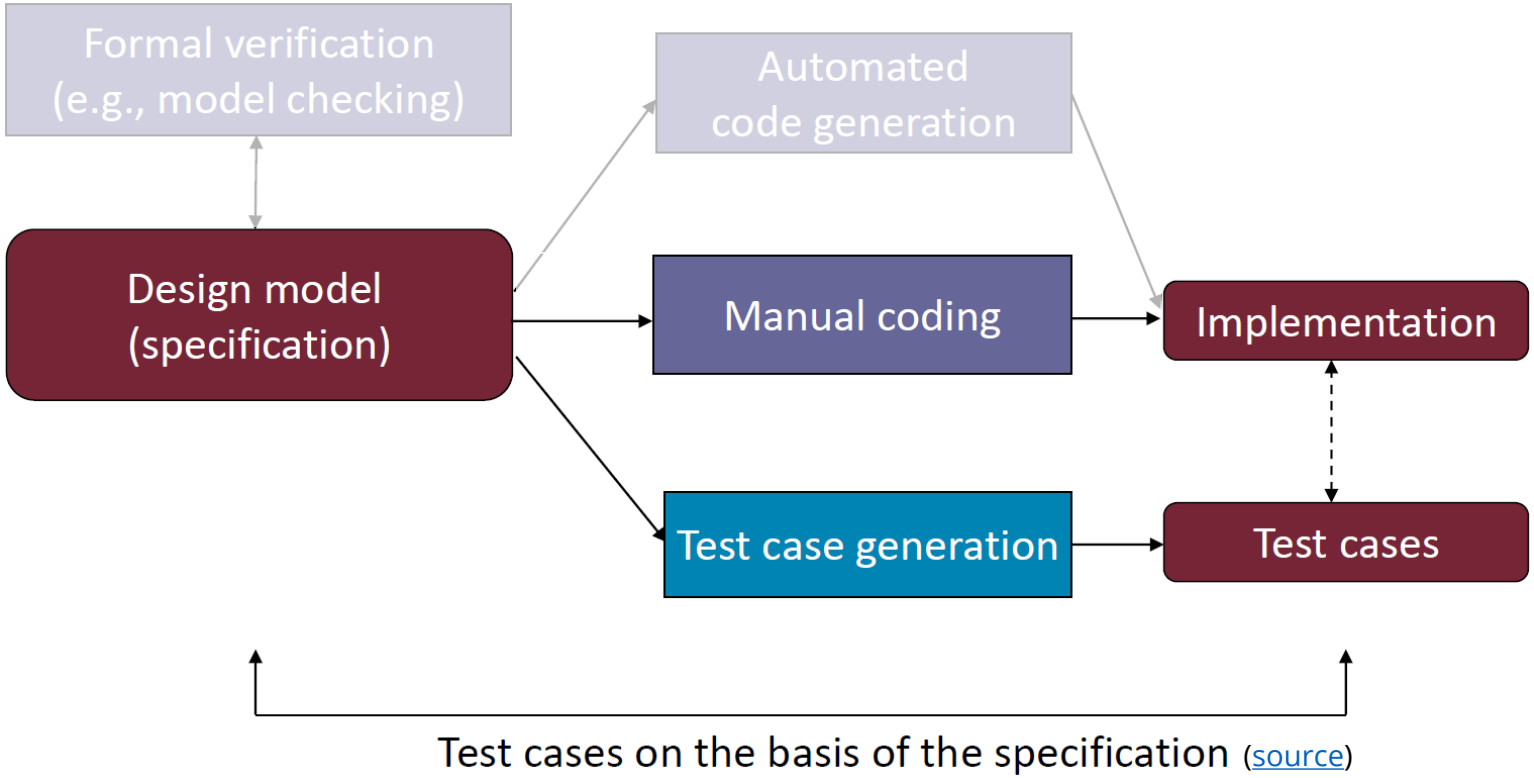


Model-Based Testing of
Read Only Graph Queries

[Lambers et al.: Model-Based Testing of Read Only Graph Queries (ICSTW'20)]

Model-Based Testing of Read Only Graph Queries

- Model-based testing: test case generation

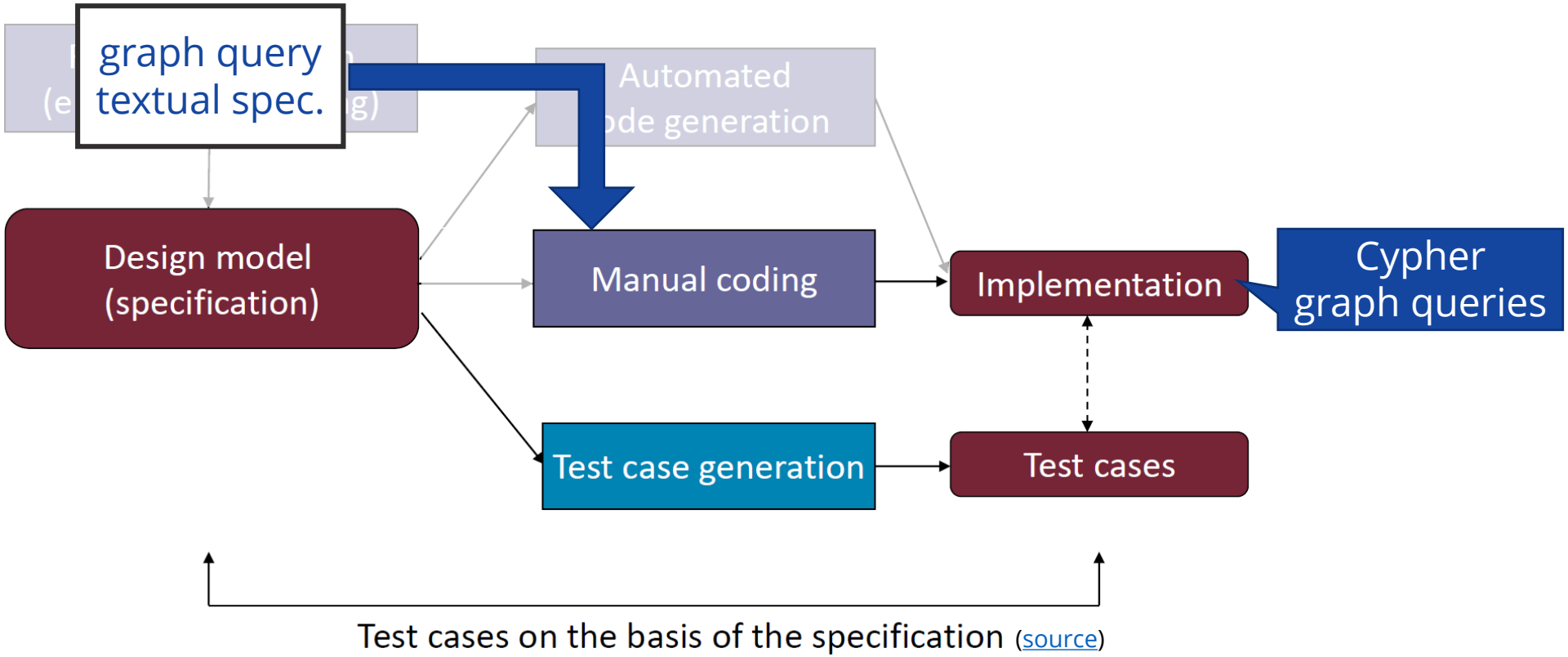


Model-Based Testing of
Read Only Graph Queries

[Lambers et al.: Model-Based Testing of Read Only Graph Queries (ICSTW'20)]

Model-Based Testing of Read Only Graph Queries

- Model-based testing: test case generation

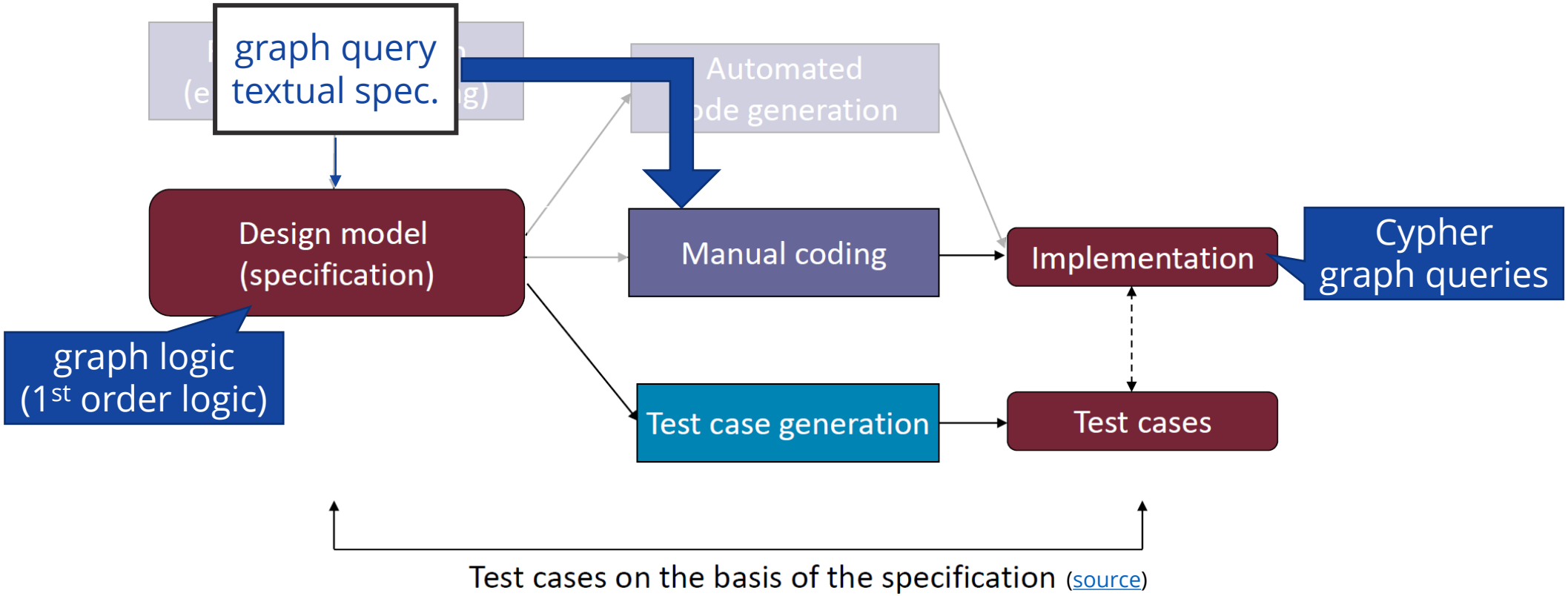


Model-Based Testing of
Read Only Graph Queries

[Lambers et al.: Model-Based Testing of Read Only Graph Queries (ICSTW'20)]

Model-Based Testing of Read Only Graph Queries

- Model-based testing: test case generation

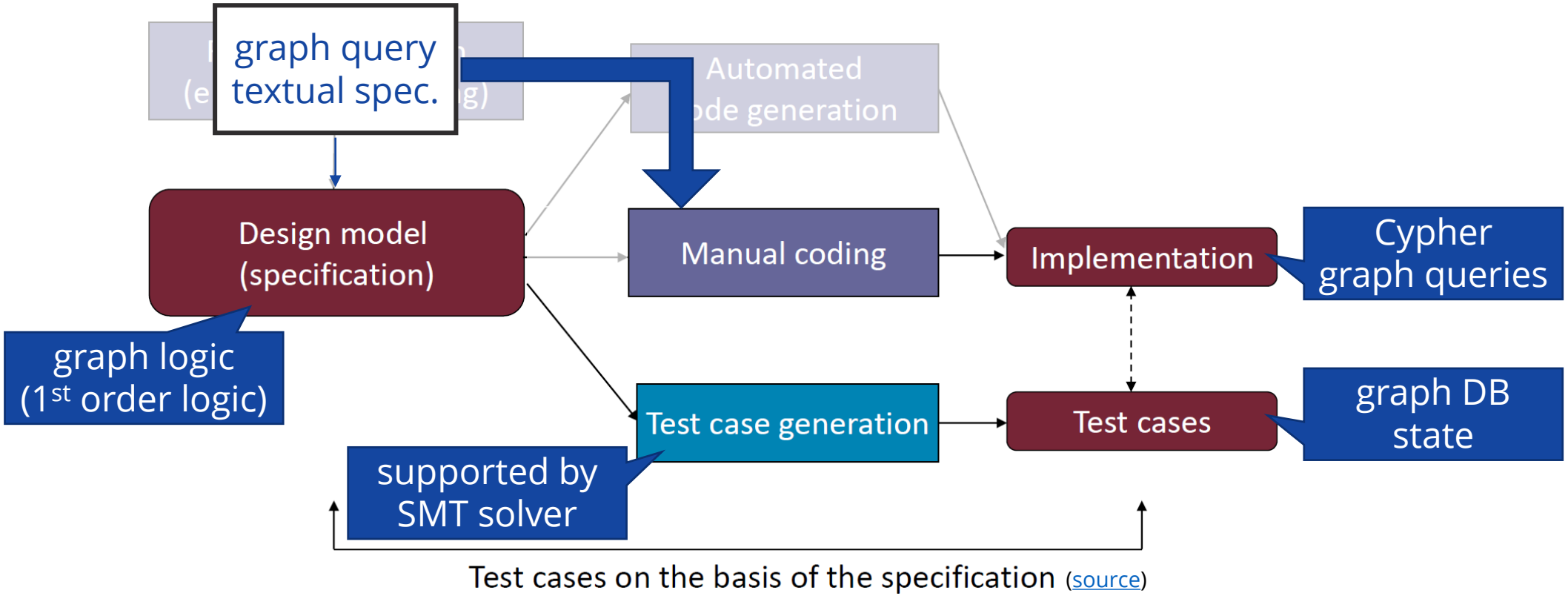


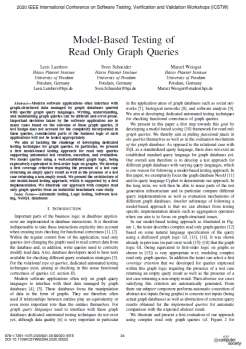
Model-Based Testing of
Read Only Graph Queries

[Lambers et al.: Model-Based Testing of Read Only Graph Queries (ICSTW'20)]

Model-Based Testing of Read Only Graph Queries

- Model-based testing: test case generation



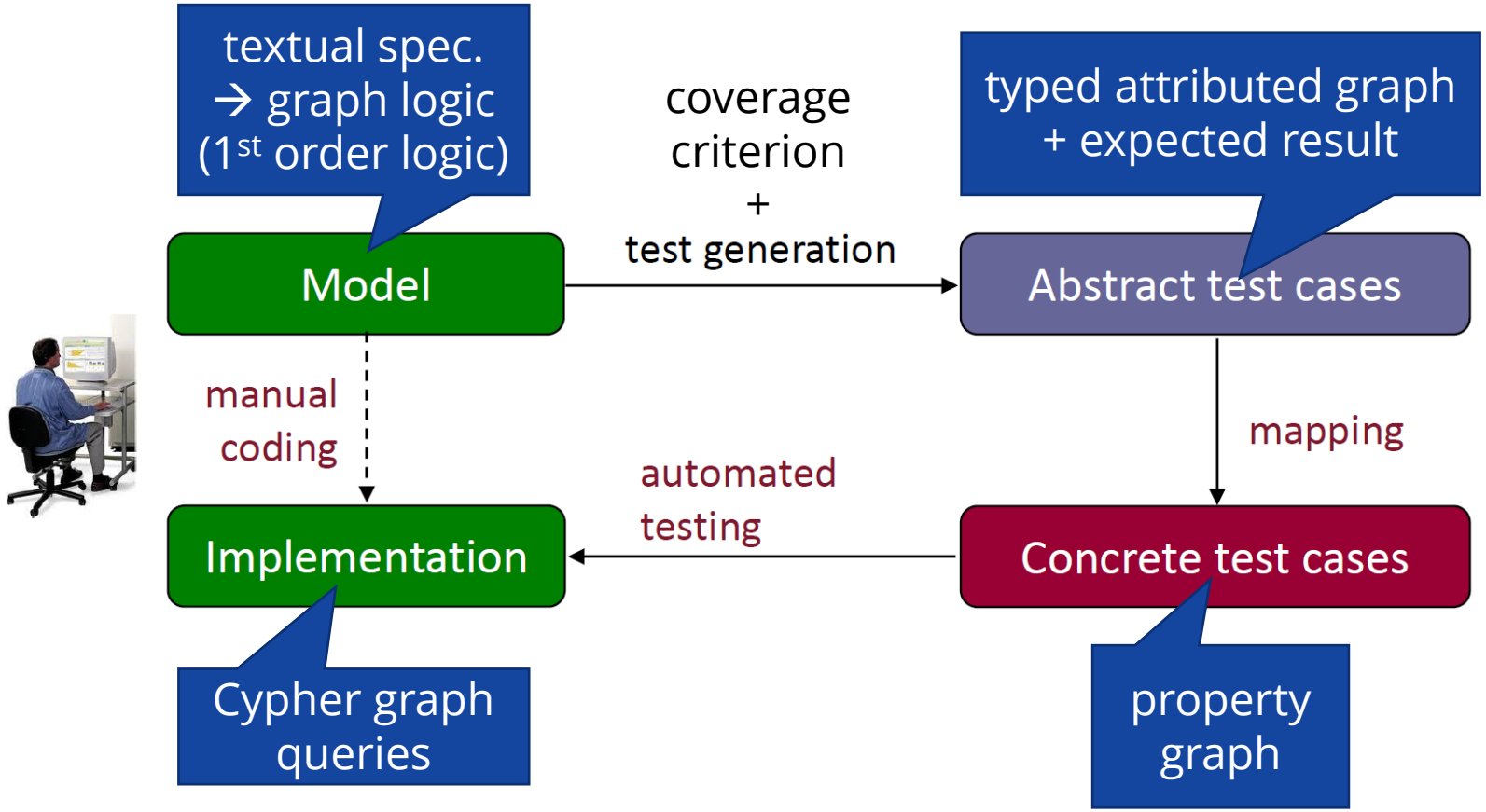


Model-Based Testing of Read Only Graph Queries

[Lambers et al.: Model-Based Testing of Read Only Graph Queries (ICSTW'20)]

Goal

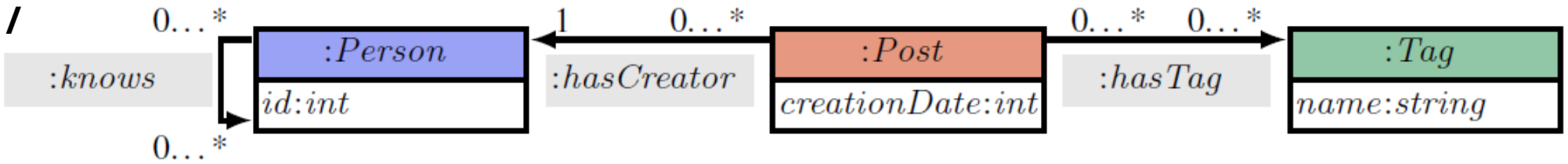
- Conformance checking



(source)

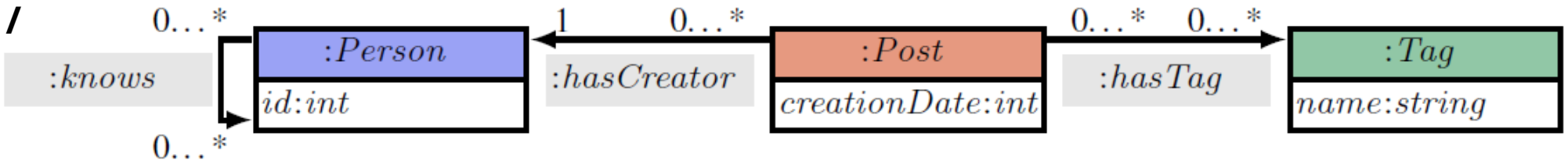
Running example

Conceptual model / schema



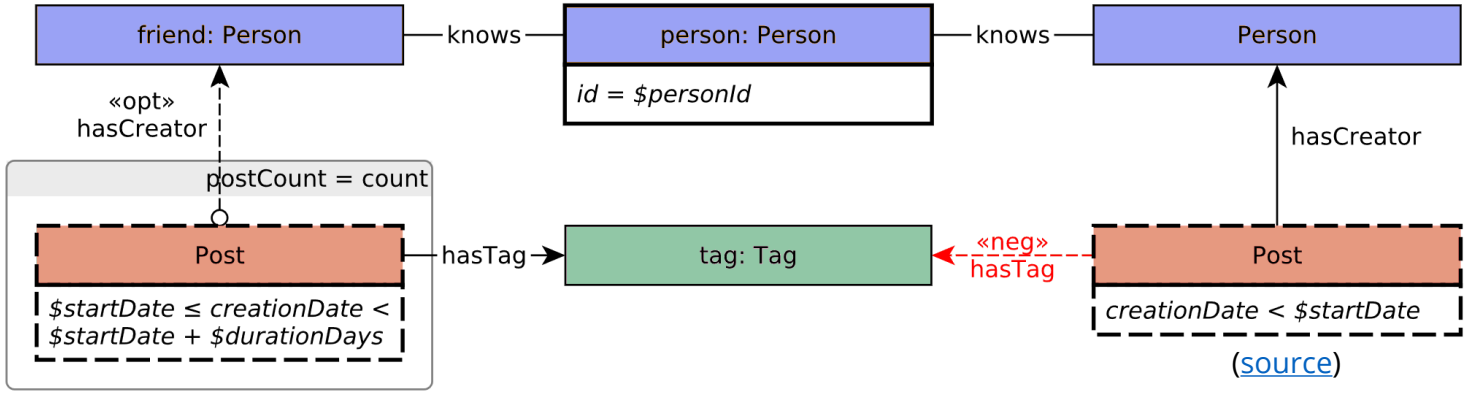
Running example

Conceptual model / schema



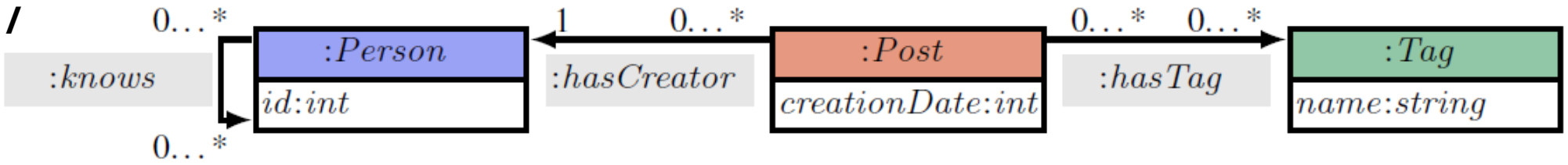
Natural language spec.: Given a start **Person** (`personId`), find **Tags** that are attached to **Posts** that were created by that **Person**'s friends. Only include **Tags** that were attached to friends' **Posts** created within a given time interval, and that were never attached to friends' **Posts** created before this interval.

(Pattern illustration)



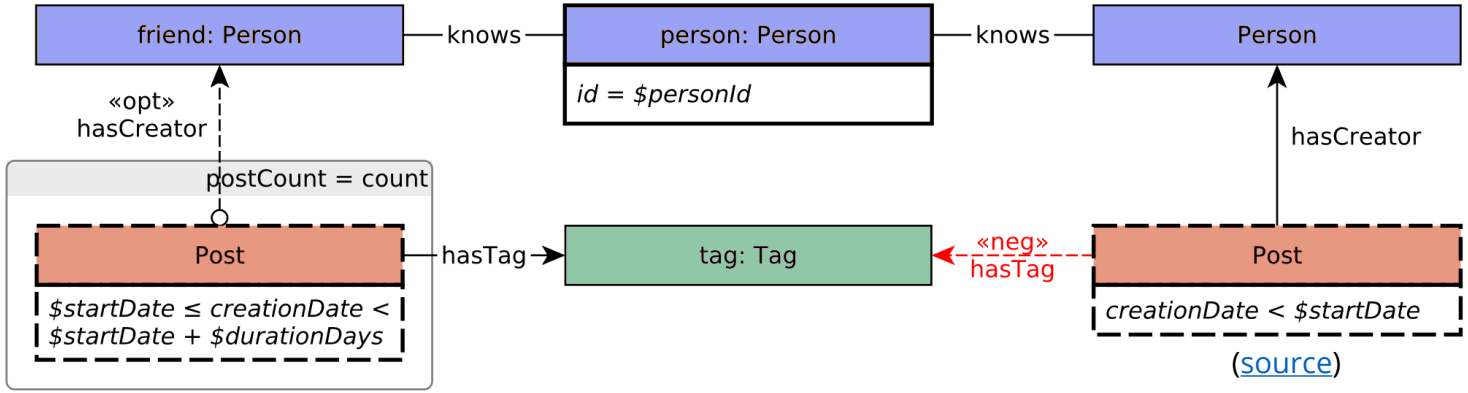
Running example

Conceptual model / schema



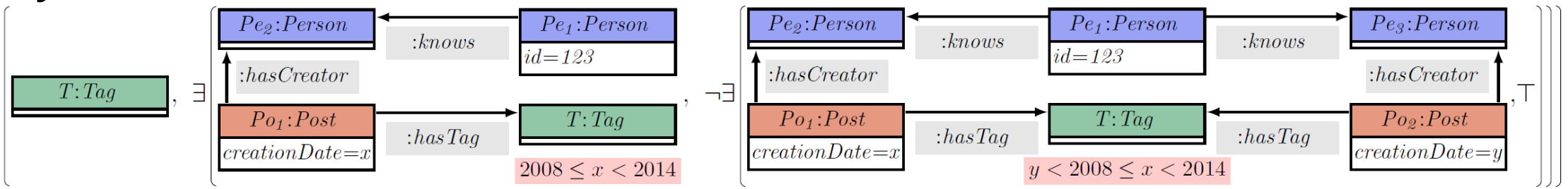
Natural language spec.: Given a start **Person** (`personId`), find **Tags** that are attached to **Posts** that were created by that **Person**'s friends. Only include **Tags** that were attached to friends' **Posts** created within a given time interval, and that were never attached to friends' **Posts** created before this interval.

(Pattern illustration)

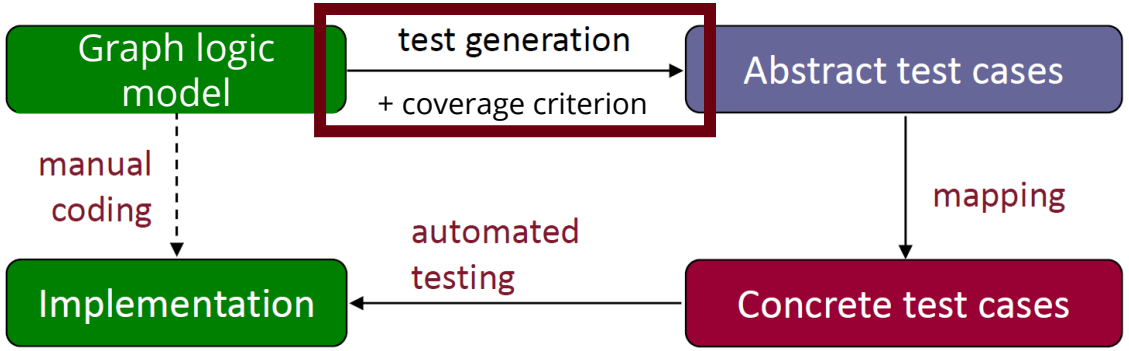


(source)

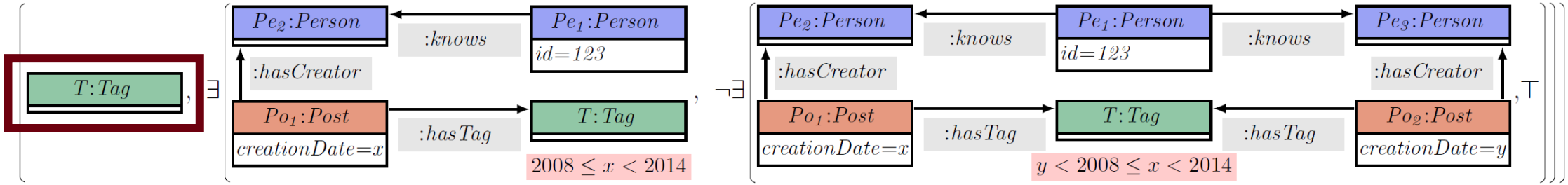
Abstract query in graph logic



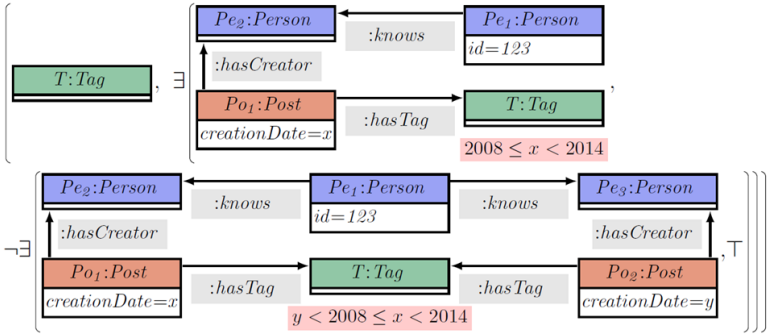
Test generation



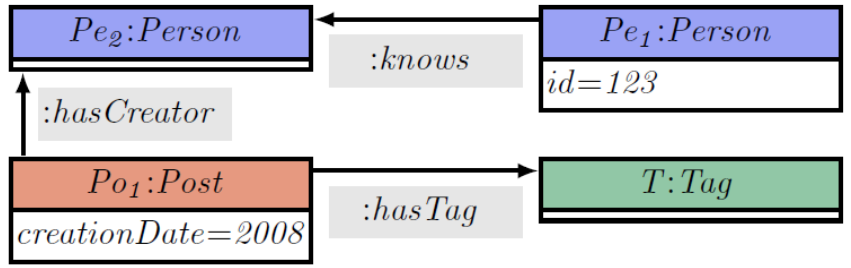
- Supported by SMT solver Z3
- Sources:
  - Domain constraints from the contextual model / schema
  - Coverage criteria:
    - predicate coverage
      - Simple criteria for the whole query
    - Test cases should return
      - Empty result
      - Non-empty result
    - To avoid oversimplified test data
      - The pattern of the result must be present in data



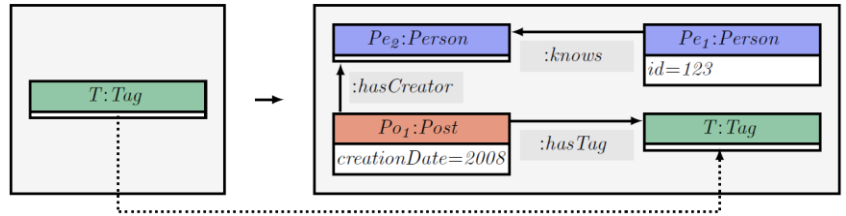
Abstraction & concretization

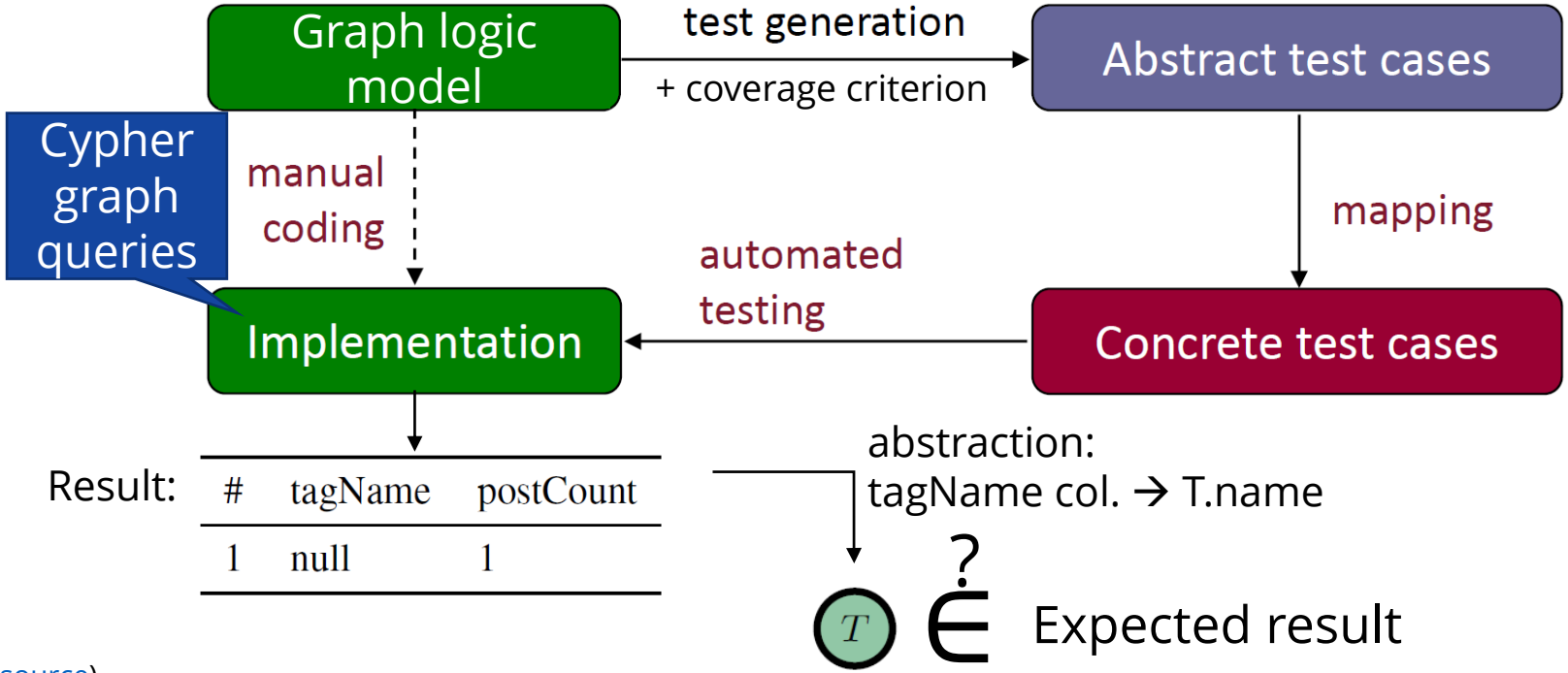


Typed attributed graph:

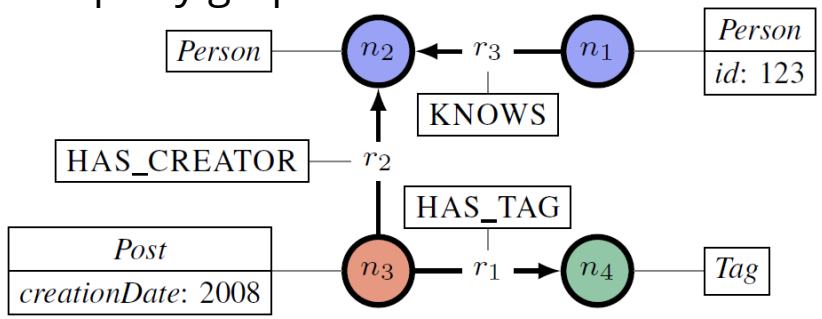


Expected result (morphism):





Property graph:



Result:

#	tagName	postCount
1	null	1

abstraction:
tagName col. → T.name

$T \overset{?}{\in}$ Expected result

(source)

Evaluation

- 4 queries from social network benchmark of Linked Data Benchmark Council (LDBC)
  - ~10 s test generation / test
  - No faulty queries or DB error found
- Ad-hoc evaluation using fault injection: mixed results

Conclusion

- Automated test generation for graph queries
  - Read-only graph queries without aggregation
  - Based on graph logic representation (manually derived)
  - Simple coverage criterion
  - Evaluated on a few LDBC graph queries
- Future work:
  - More complex coverage criteria
  - More complex read and also writing queries
  - Mutation testing
  - Embed into application test generation process
  - Mining existing queries or natural language spec. → regression testing