Software Quality for the Semantic Web

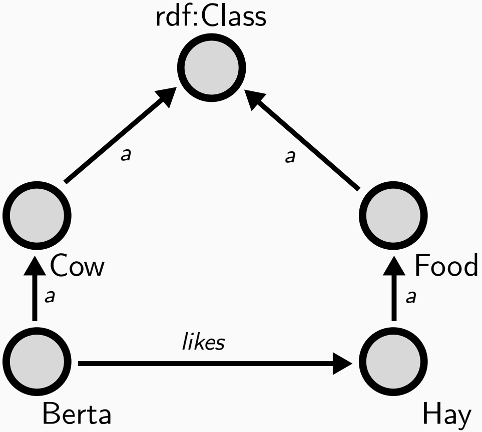
Eduard Kamburjan

Based on work with many collaborators: Tobias John, Einar Broch Johnsen, Dominic Steinhöfel, David Chaves Fraga, Romana Pernisch, Oscar Corcho, . . .

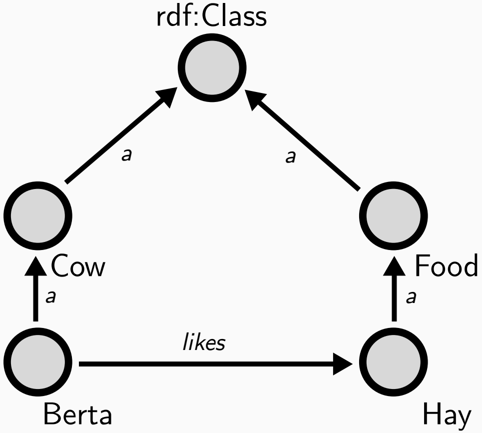
IT UNIVERSITY OF COPENHAGEN

onto:NEXUS Workshop 05.10.2025

What is a Knowledge Graph, and why do we care?



What is a Knowledge Graph, and why do we care?

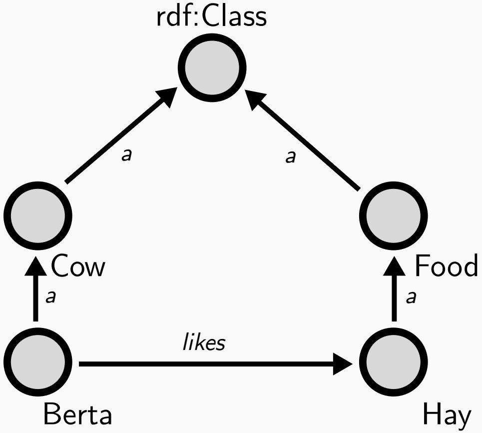


The Promise of Knowledge Graphs

A Knowledge Graph is a graph that provides high-quality semantic data to users.

- Neuro-symbolic AI: context and taming hallucinations in GenAI, ...
- Data integration: Data of high quality with agreed upon semantics, ...
- Engineering: Connecting system models with data, ...

What is a Knowledge Graph, and why do we care?



The Promise of Knowledge Graphs

A Knowledge Graph is a graph that provides high-quality semantic data to users.

- Neuro-symbolic AI: context and taming hallucinations in GenAI, ...
- Data integration: Data of high quality with agreed upon semantics, ...
- Engineering: Connecting system models with data, ...

If the central promise is quality, how do we ensure it?

What is a Knowledge Graph really?

Data Quality

Software Quality

What is a Knowledge Graph really?

Data Quality

- Data engineering pipelines with numerous tools on general data quality
- Graph specific technologies: SHACL shapes, SPARQL queries as constraints
- Formal semantics and reasoners
- Lot of different methodologies

Software Quality

What is a Knowledge Graph really?

Data Quality

- Data engineering pipelines with numerous tools on general data quality
- Graph specific technologies: SHACL shapes, SPARQL queries as constraints
- Formal semantics and reasoners
- Lot of different methodologies

Software Quality

- What about all these tools?
- What about all these pipelines?

Software Turtles all the way down

Knowledge Graphs

- A KG is a data set, generated by a set of interacting software components.
- The quality of the KG is determined also by their software quality.

Software Turtles all the way down

Knowledge Graphs

- A KG is a data set, generated by a set of interacting software components.
- The quality of the KG is determined also by their software quality.

Software Quality is Important

- Five papers in high-impact venues (including nature) retracted after a bug in python implementation of analysis algorithm

  [Miller, *Software problem leads to five retractions.*, 2007]

- Faulty analysis leads to wrong data basis for decision about austerity in Europe

  [Herndon et al., *Does High Public Debt Consistently Stifle Economic Growth? A Critique of Reinhart and Rogoff*, 2013]

- "Replication crisis" w.r.t. Jupyter notebooks: less than 25% are runnable

  [Pimentel et al., *Understanding and improving the quality and reproducibility of Jupyter notebooks*, 2021]

Agenda

- Testing software for knowledge graphs (x2)

- Dependency analysis for knowledge graph construction

A very short primer on knowledge graphs

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

A very short primer on knowledge graphs

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

A very short primer on knowledge graphs

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

RDF:

```
Paul a Person.  Peter a Person. Maria a Person.
Paul hasChild Peter. Peter hasChild Maria.
```

A very short primer on knowledge graphs

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

RDF:

```
Paul a Person.  Peter a Person. Maria a Person.
Paul hasChild Peter. Peter hasChild Maria.
```

OWL:

$\texttt{hasChild}$ some ($\texttt{hasChild}$ some $\texttt{Person}$) subClassOf $\texttt{GrandParent}$

$\exists \texttt{hasChild}.\, \exists \texttt{hasChild}.\, \texttt{Person} \sqsubseteq \texttt{GrandParent}$

A very short primer on knowledge graphs

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

RDF:

```
Paul a Person.  Peter a Person. Maria a Person.
Paul hasChild Peter. Peter hasChild Maria.
```

OWL:

$\texttt{hasChild}$ some ($\texttt{hasChild}$ some $\texttt{Person}$) subClassOf $\texttt{GrandParent}$

$\exists\texttt{hasChild}.\exists\texttt{hasChild}.\texttt{Person} \sqsubseteq \texttt{GrandParent}$

SPARQL:

```
SELECT ?x WHERE { ?x a GrandParent }
```

Testing (I): Language-based Fuzzing

Automated Testing

Problems

- How can we automatically test the general purpose tools in OE?
- How can we test the integration of ontologies with other software?
- How can we specify the integration of ontologies with other software?

- Solver and database engines are hard to test in general
- RDF has per se very little structure to constraint input generation

Automated Testing

- Language-based approach to generate random graphs and ontologies with ISLa
- Two grammars: RDF/TTL and OWL functional syntax

```
<ontology>        ::= "Ontology (" <declarations> " " <axioms> ")"
<axioms>          ::= <axiom> | <axiom> "\n" <axioms>
<axiom>           ::= <classAxiom> | <assertion> | <dataTypeDefinition> | [...]
[...]
<literal>         ::= <typedLiteral> | <stringNoLang> | <stringWithLang>
<stringWithLang> ::= <QuotedString> <LanguageTag>
```

Targets

- RDF/TTL parser and frontend utilities of Apache Jena and OWL-API
- Three OWL-EL reasoners via differential testing

[John, Kamburjan et al. *Language-Based Testing of Knowledge Graphs*, ESWC'25]

Automated Testing: Frontend Bugs

- First bug found in RDF 1.2 TTL standard with second generated file

```
<P:A> <B> <C>.
@prefix P: <http://test.no#>
```

- Both parser have bugs in corner cases, despite a formal grammar in the standard!

```
<A> <B> -.7 . // fails to parse literal
<A> <B> ; ; . // fails to parse double empty list
```

- OWL-API profile checker rejects all OWL-EL ontologies that use language tags

```
<A> <B> "test"^^xsd:String@dk_DK
```

Automated Testing: OWL-EL Reasoners

Test Targets

Three reasoners included by default with Protege

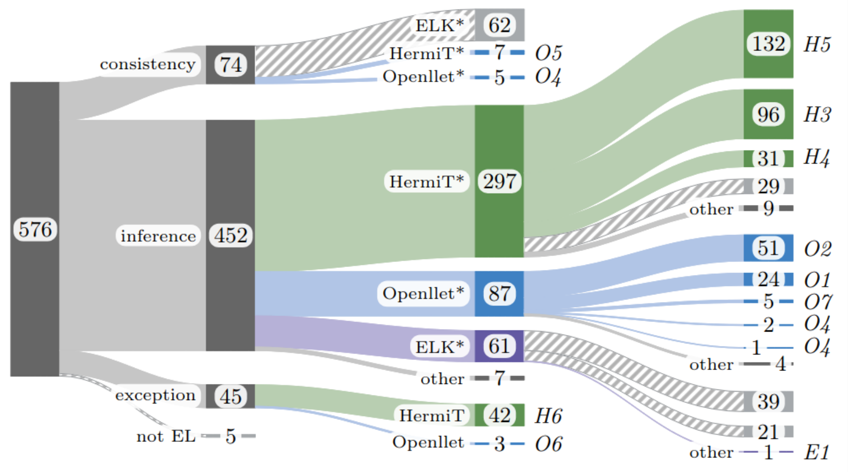
- HermiT (v.1.4.5.519)
- Pellet/Openllet (v.2.6.5)
- ELK (v.0.6.0)

Test Procedure

- Generate new ontology, and ask all three reasoners if it is consistent and to derive all possible axioms
- If results are different (or exception is thrown), investigate
- Extra tool to reduce ontology by axiom pinpointing

Automated Testing: OWL-EL Reasoners

- Found and reported 15 bugs, 13 from failed logical inference, 2 from exceptions
- Language tags and corner cases in the hierarchy



Automated Testing: OWL-EL Reasoners

```
//ELK classifies as inconsistent
Prefix(:=<http://www.example.org/reasonerTester/>)
Ontology (
    Declaration(Class(:B))   Declaration(Class(:A))
    Declaration(DataProperty(:dr))   Declaration(NamedIndividual(:a))
    EquivalentClasses( DataHasValue(:dr "s1"@fr)   :A   :B )
    DisjointClasses(   DataHasValue(:dr "s1"@en)   :A )
    ClassAssertion(:B :a))
```

```
//HermiT fails to derive DataPropertyAssertion(:dp :a "data")
Prefix(:=<http://www.example.org/reasonerTester/>)
Ontology (
    Declaration(DataProperty(:dp))  Declaration(NamedIndividual(:a))
    EquivalentClasses( ObjectOneOf(:a)  DataHasValue(:dp "data") ))
```

Automated Testing: Applicability

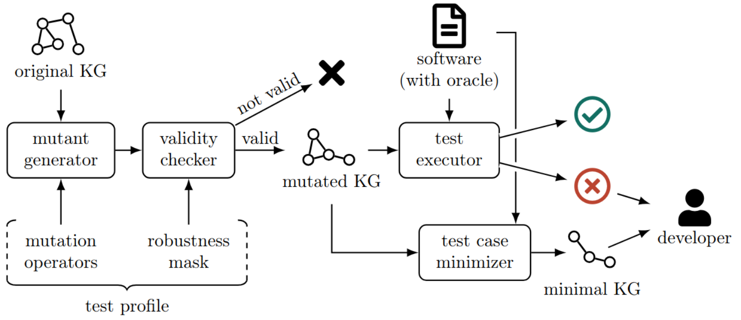
- Found numerous bugs in all tested tools, only with black box testing and limited tasks/oracles
- Proves that automated testing of general purpose tools for KGs is possible and feasible
- General purpose grammars a bit unhandy and need to be constraint by hand for more specific applications
- ISLa not optimal for high-volume generation

Testing (II): Mutation-based Integration Testing

ITU

Integration Testing

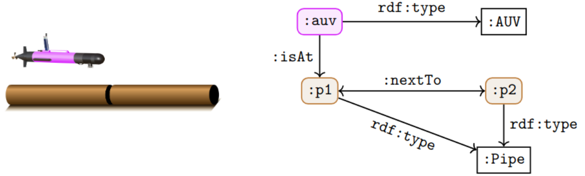
- Given a program, we often have an example KG it interacts with
- What exactly do we need to specify the program-KG interface?
- Mutation of KG to generate new inputs to program
- Challenges: Mutating KG depends on domain, program has implicit assumptions



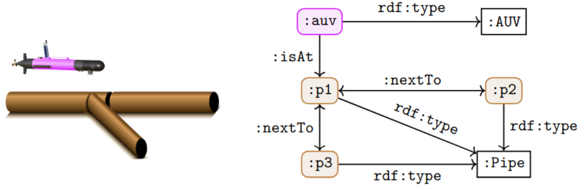
[John, Kamburjan et al. *Mutation-Based Integration Testing of Knowledge Graph Applications*, ISSRE'24]

Integration Testing: Mutation Operators

- Prior work mutate single triples or axioms
- Too fine-grained for programs – removing one entity may change a whole sub graph



(c) Scenario with corresponding KG representation before mutating.



(d) Scenario with corresponding KG representation after mutating.

Integration Testing: Robustness Mask

Robustness Mask

- Not every consistent ontology is valid input
- Program has implicit assumptions about ontology
- Top-level ontology should probably not be mutated
- Additional SHACL shapes to constrain mutations

```
p := query(":isAt(:auv, ?p)")
inspect(p)
S := query(":nextTo(p, ?s)")
while S ≠ ∅ do
    p := S.pop()
    if ¬inspected(p) then
        moveTo(p)
        inspect(p)
        S := query(":nextTo(p, ?s)")
    end if
end while
```

```
AuvAtPipeline
  a sh:NodeShape ;
  sh:targetNode :auv ;
  sh:property [
    sh:path :isAt ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:class :Pipe ;
  ].
```

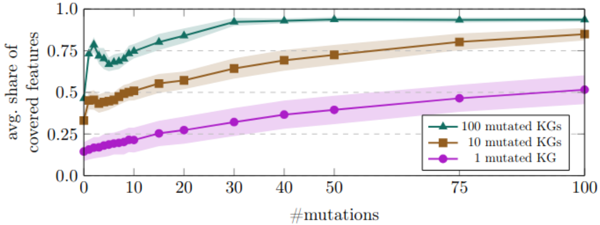
Domain-Specific Operators

- Defined per ontology or test suite
- Either directly implemented on KG (imported via Kotlin)
- Or by using SWRL syntax for rewriting

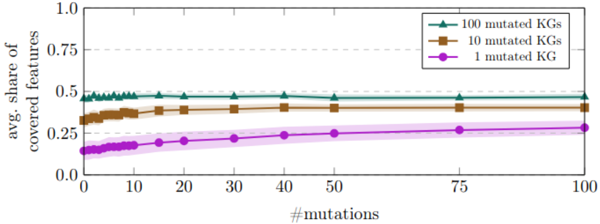
$$\texttt{rdfmutate:newNode}(?\texttt{p}) \wedge \texttt{:Topping}(?\texttt{t}) \rightarrow \texttt{:Pizza}(?\texttt{p}) \wedge \texttt{:hasTopping}(?\texttt{p}, ?\texttt{t})$$

- 59 relatively generic operators predefined
- Prototypical implementation based on rule-mining can automate initial domain-specific operators

Integration Testing: Input Coverage



(a) Domain-specific mutation operators



(b) Learned operators

- Input feature coverage: How many features are used?
- Measured via OWL vocabulary
- Domain-specific operators can be used to force feature interactions

Integration Testing: Results

Targets

- SUAVE: Simulator for self-adaptive AUV based on ROS
- GeoSimulator: Simulator for geological process based on geological ontologies
- OWL-EL reasoners: Same setup

Seed Ontologies

- Suave and GeoSimulator: Only one ontology as default example
- OWL-EL reasoners: 307 Ontologies from latest OWL reasoning competition

Results

- SUAVE: Mistakes in OWL modeling
- GeoSimulator: No bugs
- OWL-EL reasoners: 6 additional bugs related to reasoning over class hierarchies

Integration Testing: Conclusion

- Robustness mask useful for interface specification
- Even with automation, domain-specific operations require some work
- But easier to control and estimate compared to grammar-based fuzzing.
- Again, found bugs in non-trivial systems

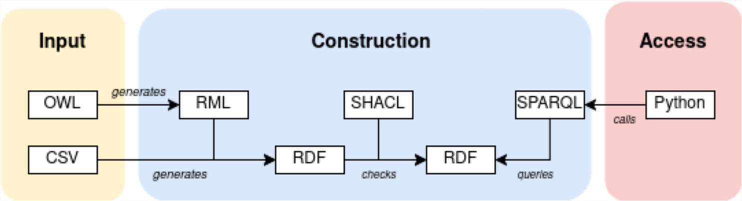
Dependency Analysis

Dependencies for KGC

Problem

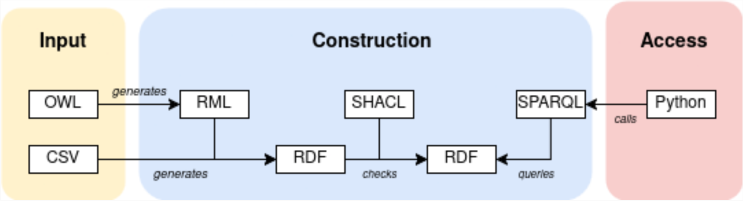
Given an KGC pipeline, can we assess the impact of a change in a component?

- Impact analysis based on a dependency analysis
- Challenge: Some used language have no formal semantics
- Challenge: Notion of dependencies not used in KGC
- First study on dependencies for impact analysis and bug detection



[Kamburjan et al. *On Dependencies in Knowledge Graph Construction*, KGWC@ESWC'25]

Dependencies



- Asset A_1 depends on asset A_2 if A_1 cannot exist without some functionality of A_2
- If A_2 changes, so must A_1.
- Explicit in programs (module systems) and software projects (gradle)
- Used for modularization, *impact analysis*, *defect analysis*

Example

Example RML:

```
roles:
  sources:
    - access: 'users.csv'
      referenceFormulation: csv
  s: dep:$(role)
  po:
    - [a, dep:Role]
    - [dep:roleName, $(role)]
```

Example SPARQL:

```
SELECT * {

?x a dep:User;
   dep:name ?name;
   dep:hasRole [dep:roleName ?roleN].

FILTER (?roleN = "Admin")
}
```

- Query depends on data output of engine driven by RML mapping
- Defect occurs, if we change URIs in the RML, but no tool can detect it!

Semantic Assets

Challenges

- Tools have no formal semantics, many domain-specific tools
- No explicit references
- Manual vs. derived assets

Internal and External Semantic Assets

- An internal semantic asset is a mapping, a graph shape or a graph query.
- An external semantic asset is input data files, ontology axioms or source code operating on the final graph

We consider mostly RML mappings, not, e.g., python mappings

External Dependencies

- A mapping M depends on a data file D, if D is input to M
- A mapping M depends on an axiom X if M is generated from X
- A program P depends on a semantic asset A, if A occurs within P

```yaml
roles:
 sources:
  - access: 'users.csv'
    referenceFormulation: csv
 s: dep:$(role)
 po:
  - [a, dep:Role]
  - [dep:roleName, $(role)]
```

Internal Dependencies

- Partial order $\preceq$ is the order of execution in the pipeline
- Library L is used to remove dependencies due to `rdf:type` etc.

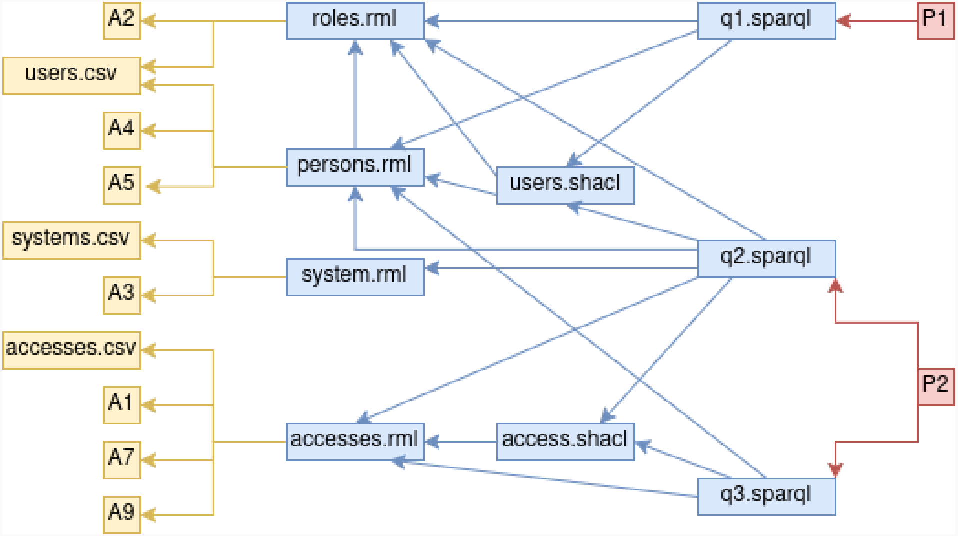
Let L $\subseteq$ URI. A semantic asset A_1 depends on another semantic asset A_2 if either

1. A_1 refers to A_2 explicitly, or
2. (2a) $A_1 \preceq A_2$, and (2b) there is some uri $\in$ L that occurs in both A_1 and A_2.

```
roles:
...
 s: dep:$(role)
 po:
  - [a, dep:Role]
  - [dep:roleName, $(role)]
```

```
SELECT * {
?x a dep:User;
   dep:name ?name;
   dep:hasRole [dep:roleName ?roleN].
FILTER (?roleN = "Admin")
}
```

Example of a Dependency Graph



Case Study: Teaching Ontology [SWJ, under review]

- 3 CSV files, 11 RML mappings, 19 SHACL shapes, 8 SPARQL Queries
- Fully automatic
- Found two bugs

Bug 1: One query without dependencies

- Accesses data using a specific URI, but the mapping was commented out.
- Maintenance bug: Corresponds to an empty test for software.

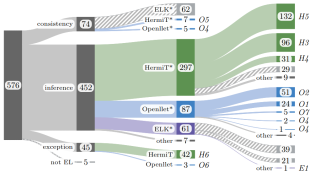
Bug 2: One shape without dependencies

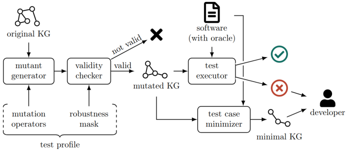
- Change of URI prefix not propagated between dependencies.
- `coursesonto:Lecturer` vs. a local URI from the developer
- Maintenance bug
- Undetected because shape validation does not fail!

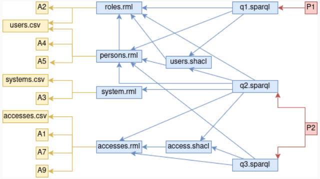
Conclusion

Conclusion

- **The Semantic Web relies on software quality**
- First steps towards investigating the field from this perspective
- Big challenges on the horizon: modularity and lack of formal semantics

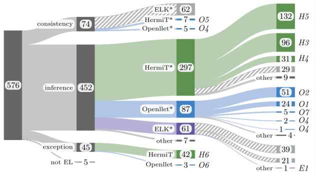


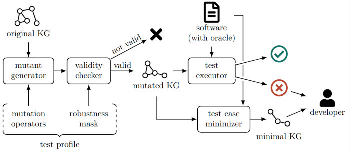


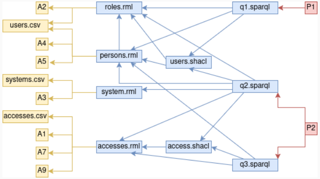


Conclusion

- **The Semantic Web relies on software quality**
- First steps towards investigating the field from this perspective
- Big challenges on the horizon: modularity and lack of formal semantics







Thank you for your attention