Shape Expression Schemas for RDF

Semantics, Complexity, and Inference

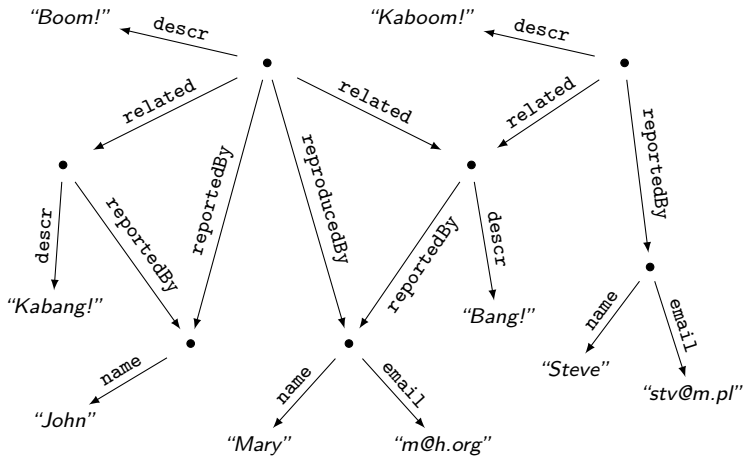
Sławek Staworko
University of Lille & INRIA LINKS

GT-ALGA 2019

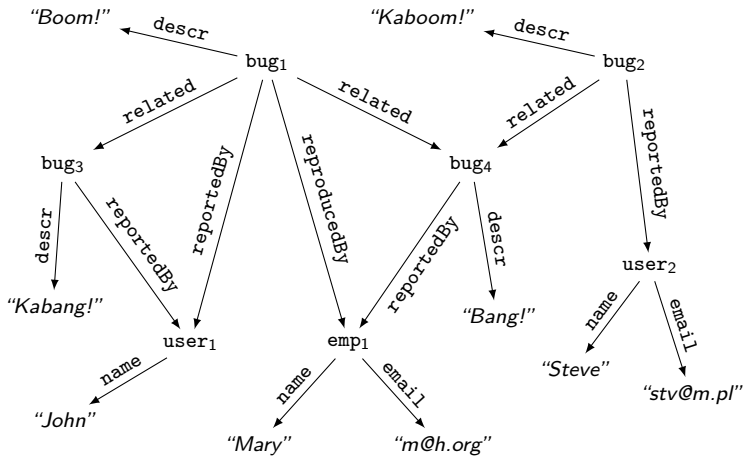
Paris, France

10 October 2019

What is RDF and does it need schemas?



What is RDF and does it need schemas?



RDF at its conception and RDF now

What is RDF?

- set of triples ⟨subject, predicate, object⟩

Originally, free-range RDF

- The *driving* technology of Web 3.0 [Tim Berners-Lee]
- *"Just publish your data so others can access it!"*
- Intentionally schema-free and ontology oriented

Nowadays, industrial-strength RDF

- Produced and consumed by applications (data exchange format)
- Often obtained from exporting data from relational databases (e.g., R2RML)
- Follows a strict structure

Outline

Uses for schemas

- Provide a semantic insight into data
- Capture the structure of the graph (summary)
- Enable validation i.e., checking data conformance

If RDF that does not come with schema?

- Infer the schema

If the RDF does not satisfy its schema?

- Repair the RDF

Initial research [ICDT'15]

- Define the semantics of ShEx
- Complexity of validation
- Expressive power

Joint work with joint work with I. Boneva, J. Labra Gayo, S. Hym, E. G. Prud'hommeaux, and H. Solbrig

Current research

- Containment problem [PODS'19]
- Grammatical inference (ongoing)

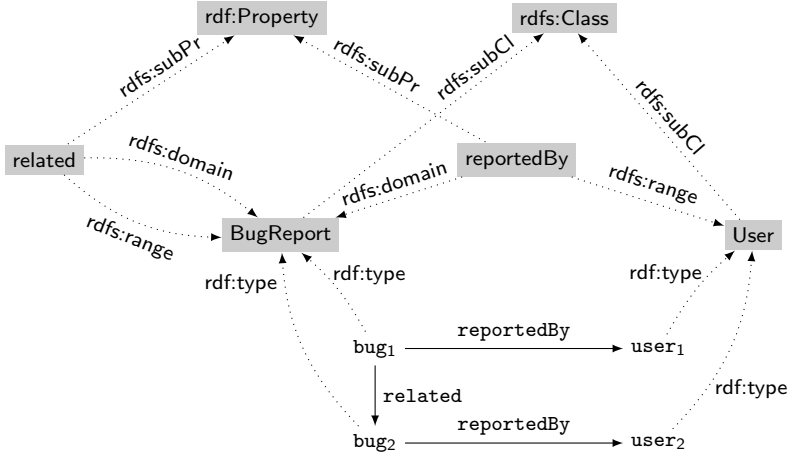
Joint work with P. Wieczorek, A. Lemay, and B. Groz

Future research

Previously existing schema formalisms for RDF

RDF Schema (RDFS) [W3C]:

- lightweight ontology language (types and type inclusion relations)
- range and domain constraints for properties (predicate types)
- virtually no power to constrain the structure of the graph



Previously existing schema formalisms for RDF (cont'd.)

OWL + CWA + UNA [Sirin, RR'10]

- Potentially confusing nonstandard semantics
- Potentially high complexity of validation

SPARQL (SPIN) [Bolleman et al., SWAT4LS'12]

- Very powerful and expressive
- High complexity

Resource Shapes [IBM, Ryman et al., LDOW'13]

- Extends RDFS with simple cardinality constraints on the outbound neighborhood of a node

What does exactly RDF validation mean

Verification the typing is given (with rdf:type) and its correctness is to be verified
(this meaning is employed by the above schema proposals)

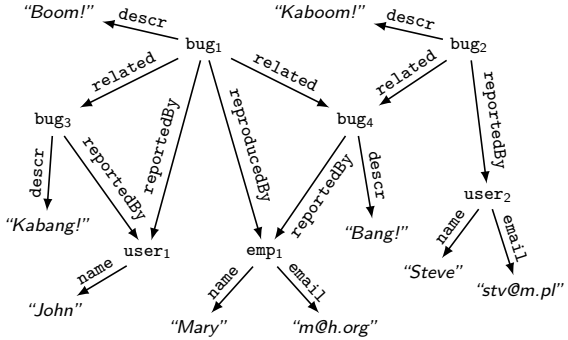
Model checking no typing is given and the goal is to construct a valid typing
(this is a more general problem that we are interested in)

Shape Expressions Schemas

Shape Expression Schema (ShEx)

Syntax

ShEx is a set of rules of the form $Type \rightarrow RegExp(Predicate \times Type)$



Bug → descr :: str,
reportedBy :: User,
reproducedBy :: Employee?,
related :: Bug*

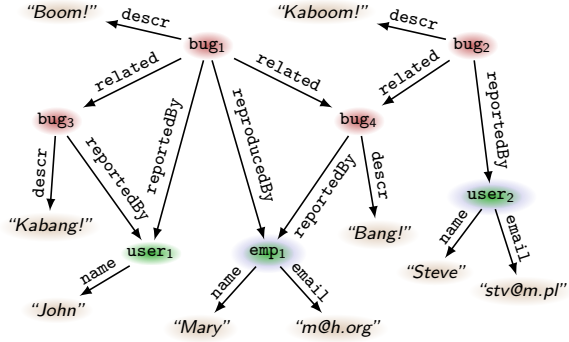
User → name :: str,
email :: str?

Employee → name :: str,
email :: str

Shape Expression Schema (ShEx)

Syntax

ShEx is a set of rules of the form $Type \rightarrow RegExp(Predicate \times Type)$




```
Bug → descr :: str,
      reportedBy :: User,
      reproducedBy :: Employee?,
      related :: Bug*
User → name :: str,
       email :: str?
Employee → name :: str,
           email :: str
```

Semantics

Graph satisfies a schema if every node has at least one type

Background information

Shape Expressions Schemas (ShEx)

- Inspired by XML Schema and reminiscent of (tree) automata
- Based on regular expressions under commutative closure
  membership NP-c [Kopczynski&To'10]; containment coNEXP-c [Haase&Hofman'16]
- Envisioned as a potential XSLT-like transformation engine for RDF

ShEx vs SHACL

- ShEx is a schema language with a growing base of users and a host of applications (e.g., Wikidata)
- SHACL is Shape Constraint Language (e.g., path constraints)
- significant overlap (upcoming paper) but also differences (recursion, negation etc.)
- comparable validation complexity (NP-complete)
- both have been developed under the tutelage of W3C
- SHACL ended up a W3C Recommendation (yay!), ShEx a W3C Community Group Project

Flavorful ShEx

```
<Emp> {
   name xsd:string | (first-name xsd:string, last-name xsd:string),
   email xsd:string,
   department @<Dept>
}

<Dept> {
   name xsd:string,
   reportedBy @<User>,
   reportedOn xsd:dateTime,
   (manager @<Emp>, appointedOn xsd:dateTime)?,
   employees @<Emp>*
}
```

Bags of symbols (unordered words)

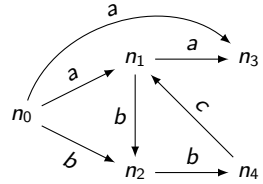
Bag (multiset) is a function mapping a symbol to the number of its occurrences.

$w_0 = \{\!|a, a, a, c, c|\!\}$ represents the function $w_0(a) = 3$, $w_0(b) = 0$, and $w_0(c) = 2$.

The collection of outgoing labels is a bag:

$$\mathit{out\text{-}lab}_G(n) = \{\!|a \mid (n, a, m) \in E_G|\!\}$$

G_0:



$\mathit{out\text{-}lab}_{G_0}(n_0) = \{\!|a, a, b|\!\}$

Bag union: $\{\!|a, c, c|\!\} \uplus \{\!|a, b|\!\} = \{\!|a, a, b, c, c|\!\}$ (concatenation of unordered words).

Regular Bag Expressions (RBEs)

Language of regular expressions for defining bags (unordered concatenation ,)

$$E ::= \epsilon \mid a \mid E^* \mid (E \text{“|”} E) \mid (E \text{“, ”} E)$$

with natural macros: $E? := (\epsilon \mid E)$ and $E^+ := (E, E^*)$

Examples

- (a^*, b^+, c, c) – arbitrary number of a's, positive number of b's, and two c's
- $(a, b)^*$ – the same number of a's and b's
- $(a, b, c)^*$ – the same number of a's, b's, and c's.

RBEs are equivalent to

1. Presburger arythmetic (PA),
2. Parikh images of context-free languages,
3. semilinear sets.

Computational properties

- Membership $w \in E$ is NP-complete,
- Emptiness $E_1 \cap E_2 = \emptyset$ is coNP-complete.

RBE_0 and $\mathrm{ShEx}_0 = \mathrm{ShEx}(\mathrm{RBE}_0)$

RBE_0

- RBEs using only symbols with multiplicities $\{0, 1, *, +, ?\}$ and , operator only
- can be canonized a, $a^? \equiv a^{[1,2]}$, b^+, $b^+ \equiv b^{[2,\infty]}$, etc.
- the canonical form is $a^{[n,n']}$, $b^{[m,m']}$, ...
- Presburger formulas: conjunctions of atoms $\#a < n$ and $\#a > n$
- captures IBM's Resource Shapes

Computational properties: simple arithmetic

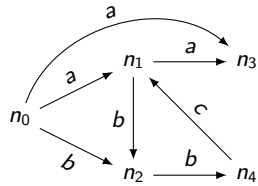
A lightweight class enjoying tractability of a number of problems:

- membership
- containment
- intersection (also with RBE_1)

Also learnable from positive examples [DBPL'13]

Getting the semantics of ShEx right

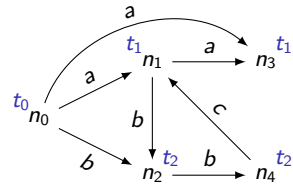
$$
\begin{aligned}
S_0\colon\quad & t_0 \to (a :: t_1)^+,\ b :: t_2 \\
& t_1 \to (a :: t_1 \mid b :: t_2)^* \\
& t_2 \to b :: t_2 \mid c :: t_1
\end{aligned}
\qquad G_0\colon
$$



Getting the semantics of ShEx right

S_0: $t_0 \to (a :: t_1)^+,\ b :: t_2$
$t_1 \to (a :: t_1 \mid b :: t_2)^*$
$t_2 \to b :: t_2 \mid c :: t_1$

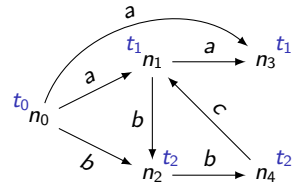
G_0:



Getting the semantics of ShEx right

S_0: $t_0 \to (a :: t_1)^+,\ b :: t_2$
$t_1 \to (a :: t_1 \mid b :: t_2)^*$
$t_2 \to b :: t_2 \mid c :: t_1$

G_0:



A single-type typing is a function $\lambda : V \to \Gamma$.

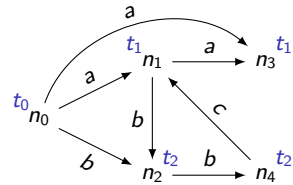
λ is valid if every node n satisfies its type definition i.e.,

$$\mathit{out\text{-}lab\text{-}type}^{\lambda}_{G}(n) = \{\!\{a :: \lambda(m) \mid (n, a, m) \in E\}\!\} \in \delta(\lambda(n)).$$

Getting the semantics of ShEx right

$$S_0: \quad t_0 \to (a :: t_1)^+,\ b :: t_2 \qquad t_1 \to (a :: t_1 \mid b :: t_2)^* \qquad t_2 \to b :: t_2 \mid c :: t_1$$

G_0:



$$\mathit{out\text{-}lab\text{-}type}^{\lambda_0}_{G_0}(n_0) = \{\!|a :: t_1, a :: t_1, b :: t_2|\!\}$$

A single-type typing is a function $\lambda : V \to \Gamma$.

λ is valid if every node n satisfies its type definition i.e.,

$$\mathit{out\text{-}lab\text{-}type}^{\lambda}_{G}(n) = \{\!|a :: \lambda(m) \mid (n, a, m) \in E|\!\} \in \delta(\lambda(n)).$$

A valid single-type typing of G_0 w.r.t. S_0

$$\lambda_0(n_0) = t_0, \qquad \lambda_0(n_1) = t_1, \qquad \lambda_0(n_2) = t_2, \qquad \lambda_0(n_3) = t_1, \qquad \lambda_0(n_4) = t_2.$$

Intractability of single-type validation

Validation problem

Checking if there exists a valid typing of given input graph w.r.t. a given input schema.

Sources of complexity

1. guessing a typing
2. checking that it is valid (RBE membership is already NP-complete)

Intractability of single-type validation

Validation problem

Checking if there exists a valid typing of given input graph w.r.t. a given input schema.

Sources of complexity

1. guessing a typing
2. checking that it is valid (RBE membership is already NP-complete)

Theorem

Single-type validation is NP-complete (even if only RBE_0 *are used).*

Reduction from graph 3-colorability:

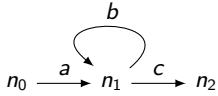
$$t_r \to \_ :: t_b^*,\ \_ :: t_g^* \qquad t_g \to \_ :: t_r^*,\ \_ :: t_b^* \qquad t_b \to \_ :: t_g^*,\ \_ :: t_r^*$$

Multi-type semantics of shape expressions

A multi-type typing is a function $\lambda : V \to 2^{\Gamma}$ that assign to every node a set of types.

S_1: $t_0 \to a :: t_1$
$t_1 \to b :: t_2,\ c :: t_3$
$t_2 \to (b :: t_2)^?,\ c :: t_3$
$t_3 \to \epsilon$

G_1:

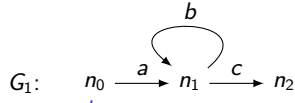


Multi-type semantics of shape expressions

A multi-type typing is a function $\lambda : V \to 2^{\Gamma}$ that assign to every node a set of types.

S_1: $t_0 \to a :: t_1$
$t_1 \to b :: t_2,\ c :: t_3$
$t_2 \to (b :: t_2)^?,\ c :: t_3$
$t_3 \to \epsilon$

G_1:

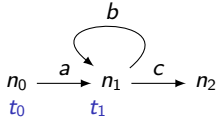


Multi-type semantics of shape expressions

A multi-type typing is a function $\lambda : V \to 2^{\Gamma}$ that assign to every node a set of types.

S_1: $t_0 \to a :: t_1$
$t_1 \to b :: t_2,\ c :: t_3$
$t_2 \to (b :: t_2)^?,\ c :: t_3$
$t_3 \to \epsilon$

G_1:

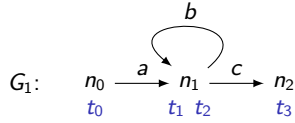


Multi-type semantics of shape expressions

A multi-type typing is a function $\lambda : V \to 2^{\Gamma}$ that assign to every node a set of types.

S_1: $t_0 \to a :: t_1$
$t_1 \to b :: t_2,\ c :: t_3$
$t_2 \to (b :: t_2)^?,\ c :: t_3$
$t_3 \to \epsilon$

G_1:

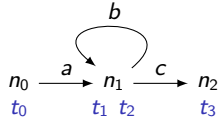


Multi-type semantics of shape expressions

A multi-type typing is a function $\lambda : V \to 2^{\Gamma}$ that assign to every node a set of types.

S_1: $t_0 \to a :: t_1$
$t_1 \to b :: t_2,\ c :: t_3$
$t_2 \to (b :: t_2)^?,\ c :: t_3$
$t_3 \to \epsilon$

G_1:



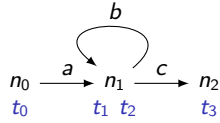
$$\lambda_1(n_0) = \{t_0\}, \qquad \lambda_1(n_1) = \{t_1, t_2\}, \qquad \lambda_1(n_2) = \{t_3\}.$$

Multi-type semantics of shape expressions

A multi-type typing is a function $\lambda : V \to 2^{\Gamma}$ that assign to every node a set of types.

S_1: $t_0 \to a :: t_1$
$t_1 \to b :: t_2,\ c :: t_3$
$t_2 \to (b :: t_2)^?,\ c :: t_3$
$t_3 \to \epsilon$

G_1:



$\lambda_1(n_0) = \{t_0\}, \qquad \lambda_1(n_1) = \{t_1, t_2\}, \qquad \lambda_1(n_2) = \{t_3\}.$

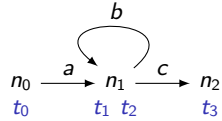
λ is valid if every node satisfies every of its associated types.

Multi-type semantics of shape expressions

A multi-type typing is a function $\lambda : V \to 2^\Gamma$ that assign to every node a set of types.

S_1: $t_0 \to a :: t_1$
$t_1 \to b :: t_2,\ c :: t_3$
$t_2 \to (b :: t_2)^?,\ c :: t_3$
$t_3 \to \epsilon$

G_1:



$\lambda_1(n_0) = \{t_0\}, \qquad \lambda_1(n_1) = \{t_1, t_2\}, \qquad \lambda_1(n_2) = \{t_3\}.$

λ is valid if every node satisfies every of its associated types.

When defining that a node n satisfies a type t...

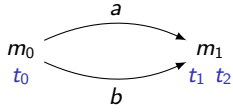
- we inspect the outbound neighborhood $\mathit{out\text{-}lab\text{-}node}_G(n) = \{(a, m) \mid (n, a, m) \in E\}$

Multi-type semantics of shape expressions

A multi-type typing is a function $\lambda : V \to 2^{\Gamma}$ that assign to every node a set of types.

$$S_2: \quad t_0 \to a :: t_1,\ b :: t_2$$
$$t_1 \to (c :: t_1)^*$$
$$t_2 \to (d :: t_2)?$$

G_2:



λ is valid if every node satisfies every of its associated types.

When defining that a node n satisfies a type t...

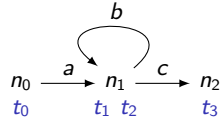
- we inspect the outbound neighborhood $\mathit{out\text{-}lab\text{-}node}_G(n) = \{(a, m) \mid (n, a, m) \in E\}$
- a node m may assume any of its assigned types $\lambda(m)$, one per edge incoming from n

Multi-type semantics of shape expressions

A multi-type typing is a function $\lambda : V \to 2^\Gamma$ that assign to every node a set of types.

S_1: $t_0 \to a :: t_1$
$t_1 \to b :: t_2,\ c :: t_3$
$t_2 \to (b :: t_2)^?,\ c :: t_3$
$t_3 \to \epsilon$

G_1:



$$\lambda_1(n_0) = \{t_0\}, \qquad \lambda_1(n_1) = \{t_1, t_2\}, \qquad \lambda_1(n_2) = \{t_3\}.$$

λ is valid if every node satisfies every of its associated types.

When defining that a node n satisfies a type t...

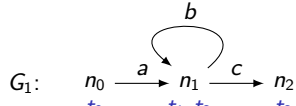
- we inspect the outbound neighborhood $\mathit{out\text{-}lab\text{-}node}_G(n) = \{(a, m) \mid (n, a, m) \in E\}$
- a node m may assume any of its assigned types $\lambda(m)$, one per edge incoming from n
- $(n, a, m) \in E$ yields the choice $|_{t \in \lambda(m)}\ a :: t$

Multi-type semantics of shape expressions

A multi-type typing is a function $\lambda : V \to 2^\Gamma$ that assign to every node a set of types.

S_1: $t_0 \to a :: t_1$
$t_1 \to b :: t_2,\ c :: t_3$
$t_2 \to (b :: t_2)^?,\ c :: t_3$
$t_3 \to \epsilon$

G_1:



$OutType(n_1, \lambda_1) = (b :: t_1 \mid b :: t_2),\ c :: t_3$

$\lambda_1(n_0) = \{t_0\}, \qquad \lambda_1(n_1) = \{t_1, t_2\}, \qquad \lambda_1(n_2) = \{t_3\}.$

λ is valid if every node satisfies every of its associated types.

When defining that a node n satisfies a type t...

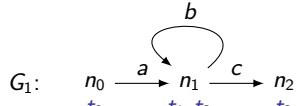
- we inspect the outbound neighborhood $out\text{-}lab\text{-}node_G(n) = \{(a, m) \mid (n, a, m) \in E\}$
- a node m may assume any of its assigned types $\lambda(m)$, one per edge incoming from n
- $(n, a, m) \in E$ yields the choice $|_{t \in \lambda(m)}\ a :: t$
- $OutType(n, \lambda) = \bigcirc_{(n,a,m) \in E}(|_{t \in \lambda(m)}\ a :: t)$

Multi-type semantics of shape expressions

A multi-type typing is a function $\lambda : V \to 2^\Gamma$ that assign to every node a set of types.

S_1: $t_0 \to a :: t_1$
$t_1 \to b :: t_2,\ c :: t_3$
$t_2 \to (b :: t_2)^?,\ c :: t_3$
$t_3 \to \epsilon$

G_1:



$OutType(n_1, \lambda_1) = (b :: t_1 \mid b :: t_2),\ c :: t_3$

$\lambda_1(n_0) = \{t_0\}, \qquad \lambda_1(n_1) = \{t_1, t_2\}, \qquad \lambda_1(n_2) = \{t_3\}.$

λ is valid if every node satisfies every of its associated types.

n satisfies t w.r.t. λ if $OutType(n, \lambda) \cap \delta(t) \neq \emptyset$,
where $OutType(n, \lambda) = \bigcirc_{(n,a,m)\in E}(|_{t\in\lambda(m)}\ a :: t)$

Refinement algorithm

The set of all valid multi-type typings of G w.r.t. S is a semi-lattice.

1. Start with the universal typing $\lambda(n) := \Gamma$
2. Iteratively refine it $\lambda := \mathit{Refine}(\lambda)$

$$[\mathit{Refine}(\lambda)](n) = \{t \in \lambda(n) \mid \mathit{OutType}(n, \lambda) \cap \delta(t) \neq \emptyset\}.$$

3. Until a fix-point is reached
4. The graph satisfies the schema iff the fix-point λ is valid ...
   ... and then λ is also the maximal valid multi-type typing.

Satisfiability of RBEs

OutType yields expressions of the form (RBE_1)

$$(a_1 \mid \cdots \mid a_k),\ \cdots,\ (z_1 \mid \ldots \mid z_m)$$

The essential decision problem for a class $\mathcal{C}$ of RBEs used in the ShEx schema is

$$\mathrm{INTER}_1(\mathcal{C}) = \{(E_0, E) \in \mathrm{RBE}_1 \times \mathcal{C} \mid E_0 \cap E \neq \emptyset\}.$$

Lemma
Tractability of INTER_1 *is a necessary and sufficient condition for tractability of multi-type validation.*

Corollary
Multi-type validation is NP-complete.

Theorem
Multi-type validation for $\mathrm{ShEx}_0 = \mathrm{ShEx}(\mathrm{RBE}_0)$ *is in PTIME*

Determinism

Determinism of shape expressions

Given the type (of a node) and the label of an outgoing edge, the expression specifies the type that the target node must satisfy.

$a :: t_1,\ b :: t_2^*,\ a :: t_1,\ c :: t_2$ deterministic

$(a :: t_1,\ b :: t_2) \mid (a :: t_3,\ c :: t_4)$ not deterministic

$a :: t_1,\ b :: t_2^*,\ a :: t_3$ not deterministic

Determinism

Determinism of shape expressions

Given the type (of a node) and the label of an outgoing edge, the expression specifies the type that the target node must satisfy.

$a :: t_1,\ b :: t_2^*,\ a :: t_1,\ c :: t_2$ $\quad$ $(a :: t_1,\ b :: t_2) \mid (a :: t_3,\ c :: t_4)$ $\quad$ $a :: t_1,\ b :: t_2^*,\ a :: t_3$

deterministic $\quad$ not deterministic $\quad$ not deterministic

Lemma

For schemas using only deterministic shape expressions, tractability of membership is a sufficient and necessary condition for tractability of multi-type validation

Proof sketch

- Knowing the label a of an outgoing edge determines the type t_a for the target node
- $\mathit{OutType}(n, \lambda) = \textcircled{,}_{(n,a,m)\in E}(|_{t\in\lambda(m)}\ a :: t)$ becomes $\textcircled{,}_{(n,a,m)\in E}(a :: t_a)$
- $\textcircled{,}_{(n,a,m)\in E}(a :: t_a)$ defines a singleton $\{w\}$ with $w = \{\!|a :: t_a \mid (n, a, m)|\!\}$
- $\mathit{OutType}(n, \lambda) \cap \delta(t) \neq \emptyset \equiv w \in \delta(t)$.

Single-occurrence REBs (SORBEs)

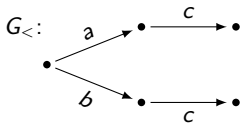
SORBE allows a symbol to be used at most once in an expression but also allows $a^{[n,m]}$

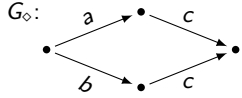
Theorem
Membership for SORBE is in PTIME :)

$a :: t_1,\ b :: t_2^*,\ a :: t_1$	$(a :: t_1,\ b :: t_2) \mid (a :: t_3,\ c :: t_4)$	$(a :: t_1,\ b :: t_2)^*,\ c :: t_3$
deterministic but not single-occurrence	not deterministic yet single-occurence	deterministic and single-occurrence

Theorem
Multi-type validation for deterministic shape expressions using SORBE *is in PTIME. :)*

Expressive power of ShEx





	FO_G	$ShEx_m$	$ShEx_s$	$\exists MSO_G$
$L : G_\diamond \in L,\ G_< \in L$	✓	✓	✓	✓
$L : G_\diamond \notin L,\ G_< \in L$	✓	×	✓	✓
$L : G_\diamond \in L,\ G_< \notin L$	✓	×	×	✓

Quick recap

Complexity

	RBE_0	RBE	SORBE	SORBE det.
multi-type	PTIME	NP-complete		PTIME
single-type	NP-complete			

Expressive power

- automata-like formalism
- incomparable with FO and MSO (unless we forbid $*$ over expressions)
- captured with MSO+PA
- incomparable with NR and HR graph grammars
- closed under intersection but not under union or negation
- single-type semantics is more expressive than multi-type semantics

Containment of ShEx

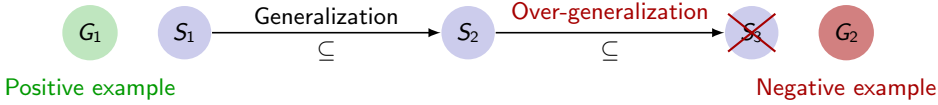
Containment problem

Containment $S_1 \subseteq S_2$

Does every graph that satisfies S_1 also satisfies S_2?

Motivation

- Fundamental problem (static analysis: query optimization, schema minimization etc.)
- Inference of ShEx (work in progress)

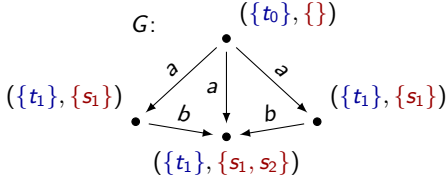


The challenge

- RBEs = Presburger Arithmetic (PA)
- $\text{MSO}_\text{G} \not\supseteq \text{ShEx} \subseteq \text{MSO}_\text{G} + \text{PA}$
- MSO_G with very little arithmetic becomes undecidable [Elgot&Rabin'66]

Decidability of Containment

$$S_1: \quad t_0 \to a :: t_1^* \qquad t_1 \to b :: t_1^? \qquad \not\subseteq \qquad S_2: \quad s_0 \to a :: s_1 \mid (a :: s_1,\ a :: s_2)^* \qquad s_1 \to b :: s_2^? \qquad s_2 \to \epsilon$$

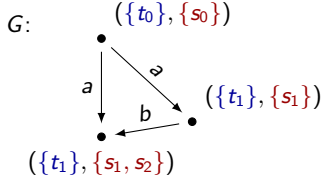


Decidability of Containment

S_1 : $t_0 \to a :: t_1^*$
$t_1 \to b :: t_1^?$

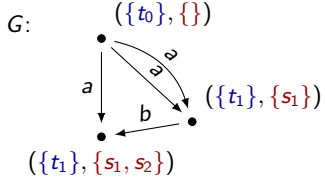
$\not\subseteq$

S_2 : $s_0 \to a :: s_1 \mid (a :: s_1,\ a :: s_2)^*$
$s_1 \to b :: s_2^?$ $\quad s_2 \to \epsilon$



Decidability of Containment

$$S_1: \quad t_0 \to a :: t_1^* \qquad t_1 \to b :: t_1^? \qquad \not\subseteq \qquad S_2: \quad s_0 \to a :: s_1 \mid (a :: s_1,\ a :: s_2)^* \qquad s_1 \to b :: s_2^? \qquad s_2 \to \epsilon$$

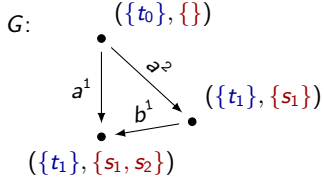


Decidability of Containment

S_1 : $t_0 \to a :: t_1^*$
$t_1 \to b :: t_1^?$

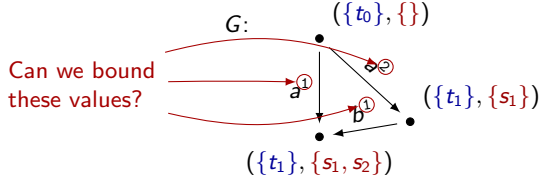
$\not\subseteq$

S_2 : $s_0 \to a :: s_1 \mid (a :: s_1,\ a :: s_2)^*$
$s_1 \to b :: s_2^?$ $\quad s_2 \to \epsilon$



Decidability of Containment

$$S_1: \quad t_0 \to a :: t_1^* \qquad t_1 \to b :: t_1^? \qquad \not\subseteq \qquad S_2: \quad s_0 \to a :: s_1 \mid (a :: s_1,\ a :: s_2)^* \qquad s_1 \to b :: s_2^? \qquad s_2 \to \epsilon$$

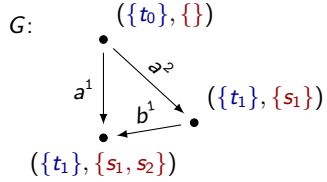


Containment of ShEx is in co2NEXP$^{\text{NP}}$

- The counter-example is a graph with at most exponential number of nodes, one node per (A, B)-kind
- There is a PA formula φ that describes the multiplicities
- PA enjoys an upper bound $O(|\varphi|^{3|\bar{x}|^k})$ on minimal solutions [Weispfenning'90]
- Double exponential upper bound on the (binary) size of the values of multiplicities
- Validation of graphs with multiplicities remains in NP

Decidability of Containment

$$S_1: \quad t_0 \to a :: t_1^* \qquad t_1 \to b :: t_1^? \qquad \not\subseteq \qquad S_2: \quad s_0 \to a :: s_1 \mid (a :: s_1,\ a :: s_2)^* \qquad s_1 \to b :: s_2^? \qquad s_2 \to \epsilon$$

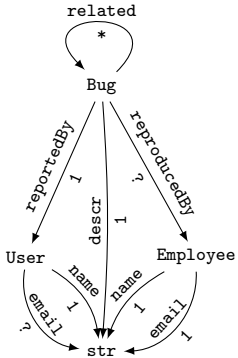


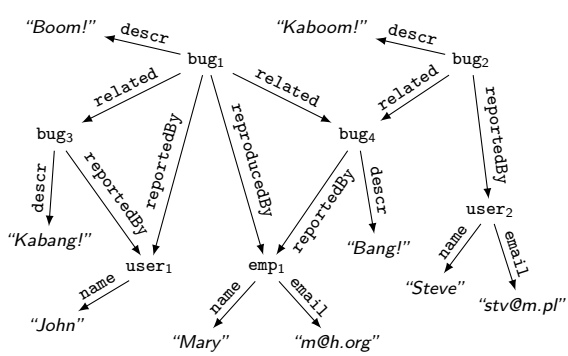
Containment of ShEx is in co2NEXP^{NP} and coNEXP-hard

- The counter-example is a graph with at most exponential number of nodes, one node per (A, B)-kind
- There is a PA formula φ that describes the multiplicities
- PA enjoys an upper bound $O(|\varphi|^{3|\bar{x}|^k})$ on minimal solutions [Weispfenning'90]
- Double exponential upper bound on the (binary) size of the values of multiplicities
- Validation of graphs with multiplicities remains in NP
- Containment of commutative REs recently shown to be coNEXP-hard [Haase&Hofman'16]

ShEx_0

- no disjunction $(a :: t_1 \mid b :: t_2)$ and no grouping $(a :: t_1, b :: t_2)^*$
- Shape Graphs – an equivalent graphical representation





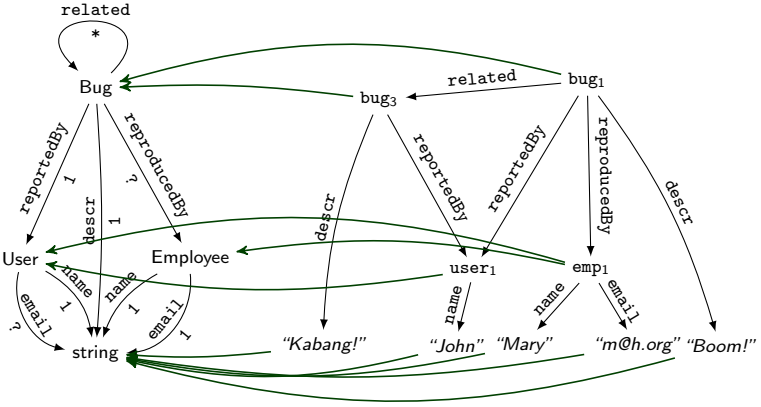
$\texttt{Bug} \rightarrow \texttt{descr} :: \texttt{str},\ \texttt{reportedBy} :: \texttt{User},\ \texttt{reproducedBy} :: \texttt{Employee}^?,\ \texttt{related} :: \texttt{Bug}^*$

$\texttt{User} \rightarrow \texttt{name} :: \texttt{str},\ \texttt{email} :: \texttt{str}^?$

$\texttt{Employee} \rightarrow \texttt{name} :: \texttt{str},\ \texttt{email} :: \texttt{str}$

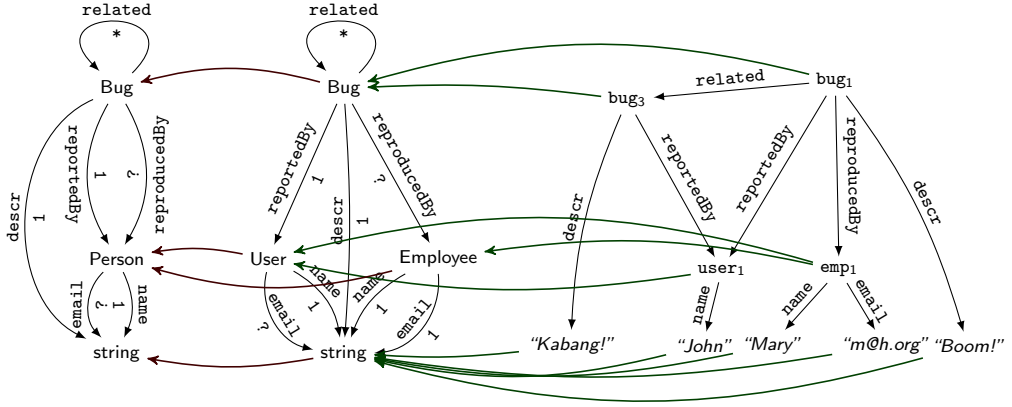
Embeddings

- Graph morphism with occurrence constraints, closely related to graph simulations
- Capture semantics of ShEx_0 by means of structural comparison



Embeddings

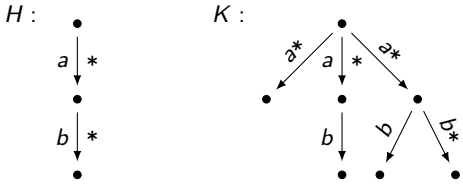
- Graph morphism with occurrence constraints, closely related to graph simulations
- Capture semantics of ShEx$_0$ by means of structural comparison
- Generalize naturally to pairs of shape graphs



Properties of embeddings

Embedding and containment

- Embedding implies containment
- In general, the converse does not hold



H cannot be embedded into K ($b :: t^*$ is equivalent to $\epsilon \mid b :: t \mid b :: t^+$)

Theorem

Constructing embeddings is

- in PTIME if only 1, ?, *, + are used
- NP-complete if arbitrary occurrence constraints are allowed $a :: t^{[n;m]}$

When does containment implies embedding?

Determinism

- DetShEx$_0$ every type uses each predicate symbol at most once
- DetShEx$_0^-$ no + are allowed and ? must be dominated by $*$

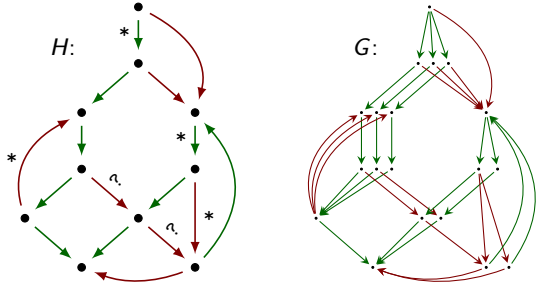
Characterizing graph

For any $H \in$ DetShEx$_0^-$ there is a polynomially-sized graph G characterizing H under containment i.e.,

$$\forall K \in \text{DetShEx}_0^-.\ G \text{ satisfies } K \Rightarrow H \subseteq K.$$

Theorem
Containment for DetShEx$_0^-$ is in PTIME

Theorem
Containment for DetShEx$_0$ is coNP-hard



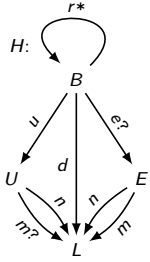
Two equivalent ShEx_0 schemas and their shape graphs

H :

$\texttt{Bug} \to \texttt{descr} :: \texttt{str},\ \texttt{reportedBy} :: \texttt{User},\ \texttt{reproducedBy} :: \texttt{Employee}^{?},$
$\texttt{related} :: \texttt{Bug}^{*}$

$\texttt{User} \to \texttt{name} :: \texttt{str},\ \texttt{email} :: \texttt{str}^{?}$

$\texttt{Employee} \to \texttt{name} :: \texttt{str},\ \texttt{email} :: \texttt{str}$



K :

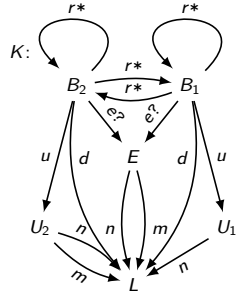
$\texttt{User}_1 \to \texttt{name} :: \texttt{str}$

$\texttt{User}_2 \to \texttt{name} :: \texttt{str},\ \texttt{email} :: \texttt{str}$

$\texttt{Bug}_1 \to \texttt{descr} :: \texttt{str},\ \texttt{reportedBy} :: \texttt{User}_1,\ \texttt{reproducedBy} :: \texttt{Employee}^{?},$
$\texttt{related} :: \texttt{Bug}_1^{*},\ \texttt{related} :: \texttt{Bug}_2^{*}$

$\texttt{Bug}_2 \to \texttt{descr} :: \texttt{str},\ \texttt{reportedBy} :: \texttt{User}_2,\ \texttt{reproducedBy} :: \texttt{Employee}^{?},$
$\texttt{related} :: \texttt{Bug}_1^{*},\ \texttt{related} :: \texttt{Bug}_2^{*}$

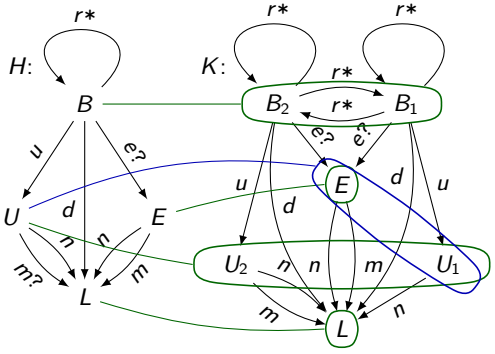
$\texttt{Employee} \to \texttt{name} :: \texttt{str},\ \texttt{email} :: \texttt{str}$



Coverings

Generalization of embeddings

A type t is covered by a set of types $S = \{s_1, \ldots, s_k\}$ iff any node satisfying t also satisfies one of the types in S

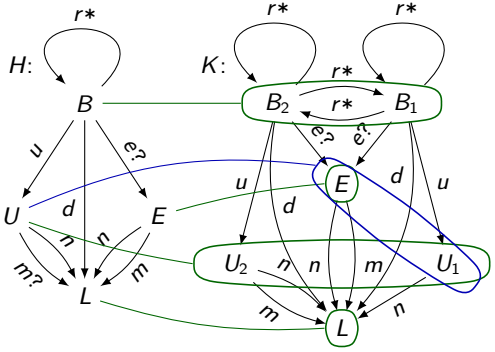


Lemma (Constructing covering)

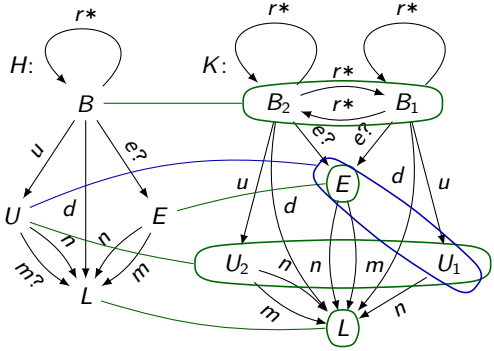
Covering is the maximum relation $R \subseteq \text{Types}(H) \times \mathcal{P}(\text{Types}(K))$ such that

$$\forall (t, S) \in R.\ \text{def}(t) \xrightarrow{\textit{Unfold}}_R \{\text{def}(s) \mid s \in S\}.$$

Unfolding



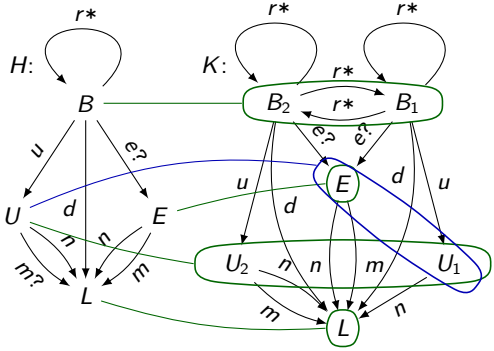
Unfolding



Unfolding U into $\{U_1, U_2\}$

$$U \to n :: L,\ m :: L^{?} \equiv n :: L,\ (\epsilon \mid m :: L) \equiv (n :: L) \mid (n :: L,\ m :: L) \leftarrow U_1 \mid U_2$$

Unfolding



Unfolding B into $\{B_1, B_2\}$

$$
\begin{aligned}
B &\rightarrow r :: B^*,\ u :: U,\ d :: L,\ e :: E^? \\
&\equiv (r :: B^*,\ u :: U_1,\ d :: L,\ e :: E^?) \mid (r :: B^*,\ u :: U_2,\ d :: L,\ e :: E^?) \\
&\equiv (r :: B_1^*,\ r :: B_2^*,\ u :: U_1,\ d :: L,\ e :: E^?) \mid (r :: B_1^*,\ r :: B_2^*,\ u :: U_2,\ d :: L,\ e :: E^?) \\
&\leftarrow B_1 \mid B_2
\end{aligned}
$$

Complexity of $ShEx_0$

Theorem
Containment for $ShEx_0$ is in EXP

- Covering is a relation of exponential size
- Covering can be obtained with an iterative refinement process
  (starting with maximal relation and remove at least one element at each iteration until stabilization)
- At each step unfoldings are constructed and each unfolding is a tree whose size is bounded exponentially

Theorem
Containment for $ShEx_0$ is EXP-complete

- Reduction from containment for binary tree automata

Quick recap

- Containment for ShEx is decidable
- There is a (arguably practical) class DetShEx$_0^-$ with tractable containment
- ShEx is very different from tree automata and requires novel techniques

ShEx	DetShEx	ShEx$_0$	DetShEx$_0$	DetShEx$_0^-$
coNEXP-h and co2EXP^{NP}	co2EXP	EXP-c	coNP-h	PTIME

Inference of ShEx

Constructing ShEx from an RDF Graph

What for?

- RDF originally schema-free but schemas are useful
- Free-range RDF is dirty
- Industrial-strength RDF is relatively clean and exhibits regular structure

$$\text{Relational database} \xrightarrow{\text{R2RDF}} \text{RDF}$$

Constructing ShEx from an RDF Graph

What for?

- RDF originally schema-free but schemas are useful
- Free-range RDF is dirty
- Industrial-strength RDF is relatively clean and exhibits regular structure

$$\text{Relational database} \xrightarrow{\text{R2RDF}} \text{RDF}$$

What do we want?

For a given RDF Graph G construct a ShEx schema S that captures the structure of G:

Soundness	G satisfies S
Succinctness	S is small enough for a human user to consume
Specificity	S is not overly general (as the universal schema is)

Implicit node similarity

Two nodes should have the same type if they are similar in some way. Two possible criteria:

- content – the outbound neighborhood is similar
- context – the inbound neighborhood is similar

Inference and Fitting

Fitting

For a input G construct $\subseteq$-minimal schema S that G satisfies.

Inference

An inference algorithm A is an algorithm that is

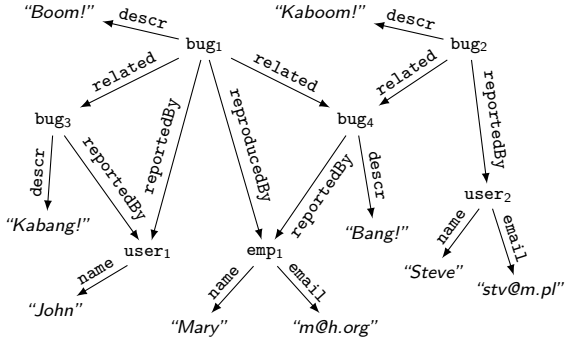
- sound i.e., it returns a schema that the input graph satisfies;
- complete i.e., it can return any goal schema provided that the input graph is sufficiently informative (or rich enough).

Both approaches are parameterized by a class $\mathcal{C}$ of goal ShEx schemas.

Bad news for Fitting: Overfitting

Fitting for ShEx_0 is trivial and potentially too verbose

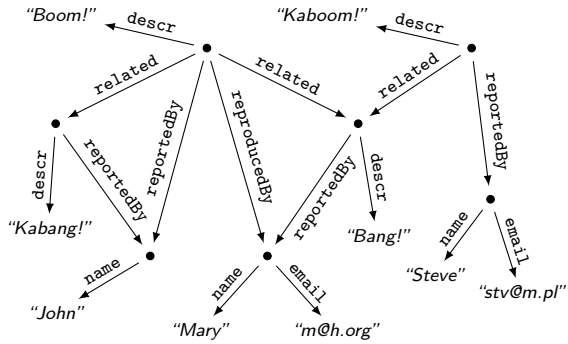
An RDF graph interpreted as a shape graph is its own ShEx_0 fitting



Bad news for Fitting: Overfitting

Fitting for ShEx_0 is trivial and potentially too verbose

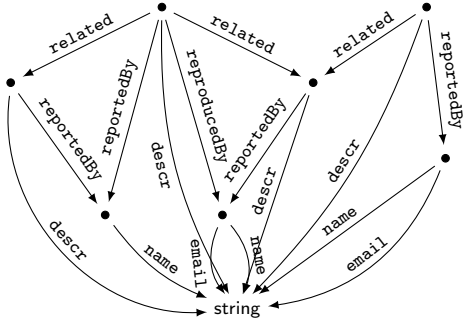
An RDF graph interpreted as a shape graph is its own ShEx_0 fitting



Bad news for Fitting: Overfitting

Fitting for ShEx_0 is trivial and potentially too verbose

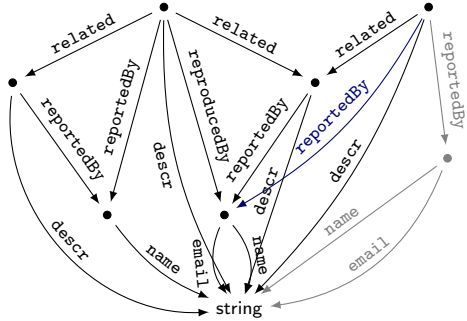
An RDF graph interpreted as a shape graph is its own ShEx_0 fitting



Bad news for Fitting: Overfitting

Fitting for ShEx_0 is trivial and potentially too verbose

An RDF graph interpreted as a shape graph is its own ShEx_0 fitting



Bisimulation can identify redundant/repetitive information

But it won't generalize the result (it won't introduce $*$ or recursion).

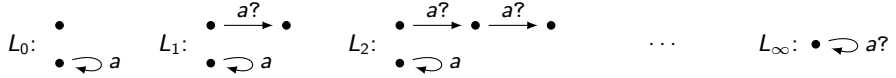
Bad news for Inference: Limit Point

Limit point

A class of languages $\mathcal{C}$ has the *limit point* property iff $\mathcal{C}$ contains an ascending chain of languages $L_1 \subsetneq L_2 \subsetneq \dots$ whose limit point $L_\infty = \bigcup_i L_i$ also belongs to $\mathcal{C}$.

Folklore result: Limit point precludes inference

No family with limit point property has an inference algorithm.



Lemma

For any $M \subseteq \{1, ?, *, +\}$ with at least two elements the class $\mathsf{ShEx}_0(M)$ has the limit point property.

So... What do we do?

We compromise

- Find a suitable subclass that allows inference and remains relatively practical.
- Motivation: first draft of a ShEx schema for an architect to work on.

So... What do we do?

We compromise

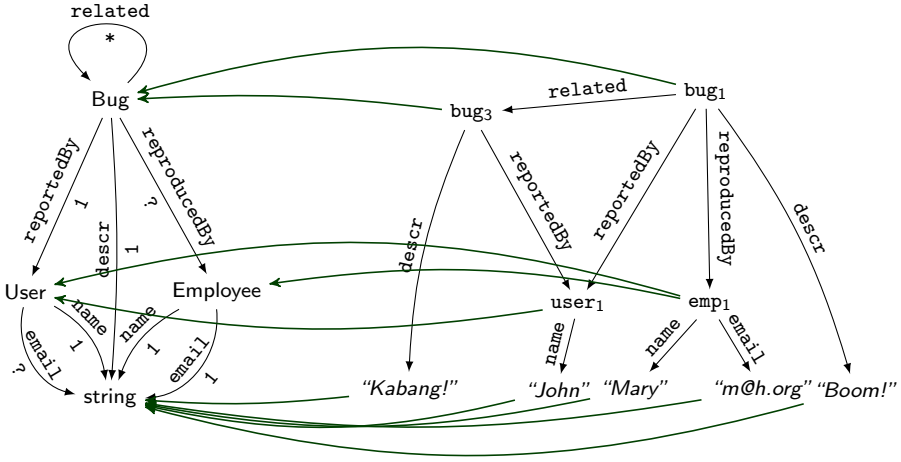
- Find a suitable subclass that allows inference and remains relatively practical.
- Motivation: first draft of a ShEx schema for an architect to work on.

Our results

	ShEx_0	$\mathrm{SingShEx}_0$	$\mathrm{DetShEx}_0$	Typed graph
Fitting	trivial	undefined (?)	exponential	NP-hard
Inference	unfeasible	$\mathrm{SingShEx}_0(1, *)$	$\mathrm{DetShEx}_0^-$	ShEx_0 (?)

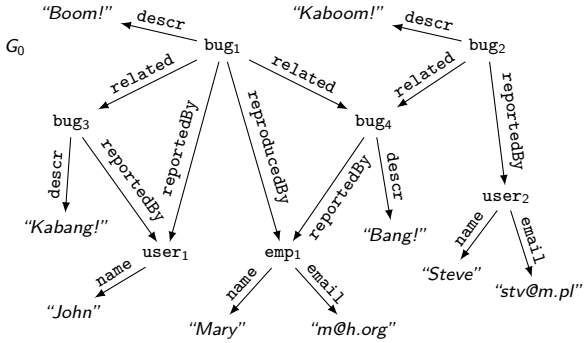
Embeddings (recall)

- Generalize simulations
- Capture semantics of ShEx by means of structural comparison

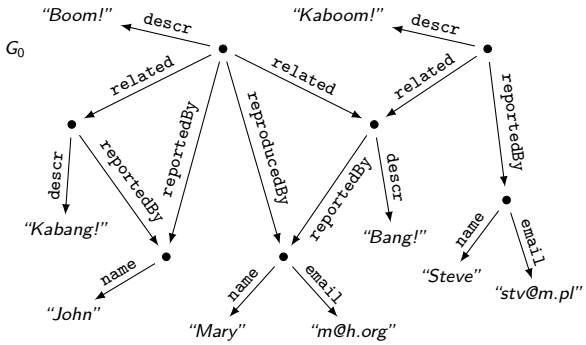


Singular Shape Expression Schemas

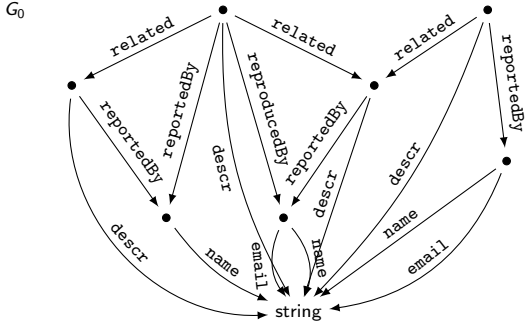
Inference of SingShEx$_0$: Generalization



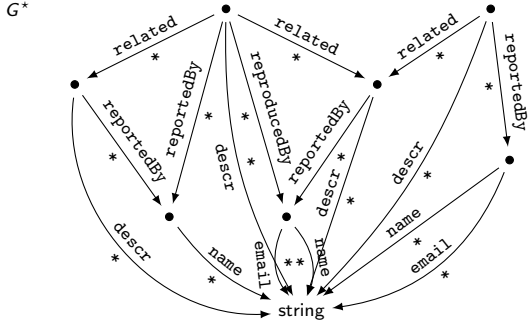
Inference of SingShEx$_0$: Generalization



Inference of SingShEx$_0$: Generalization

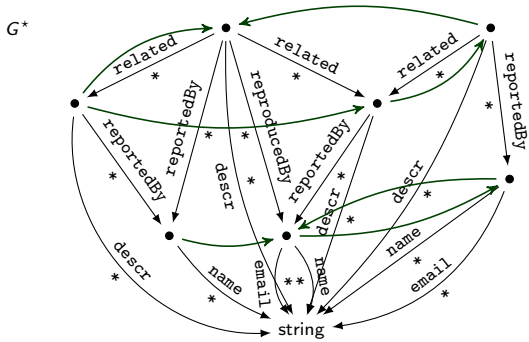


Inference of SingShEx$_0$: Generalization



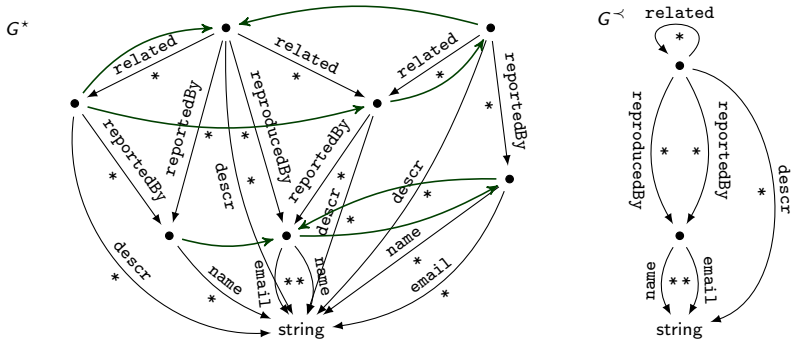
1. Put $*$ on every edge of the input graph

Inference of SingShEx$_0$: Reduction



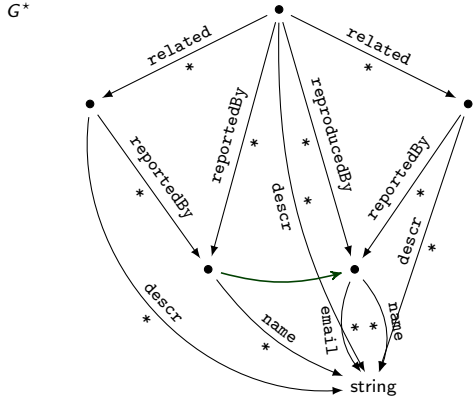
1. Put $*$ on every edge of the input graph
2. Construct the autoembedding

Inference of SingShEx$_0$: Reduction



1. Put $*$ on every edge of the input graph
2. Construct the autoembedding
3. Remove any dominated nodes

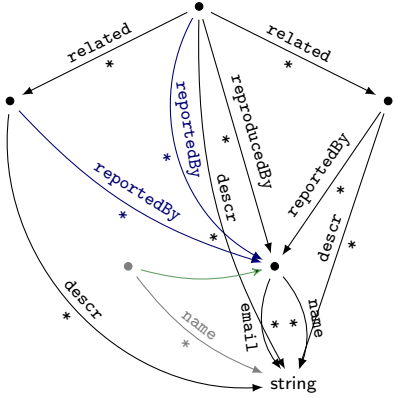
Inference of SingShEx$_0$: Reduction in detail



Removing a node n dominated by m

- remove n and all of its outgoing edges
- redirect its incoming edges to m

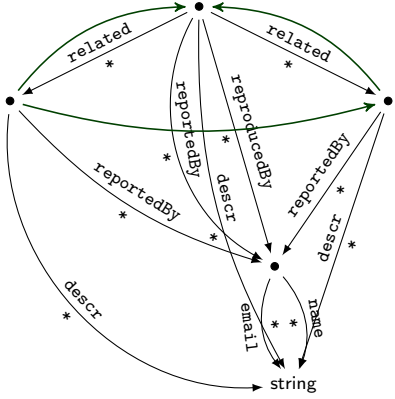
Inference of SingShEx$_0$: Reduction in detail



Removing a node n dominated by m

- remove n and all of its outgoing edges
- redirect its incoming edges to m

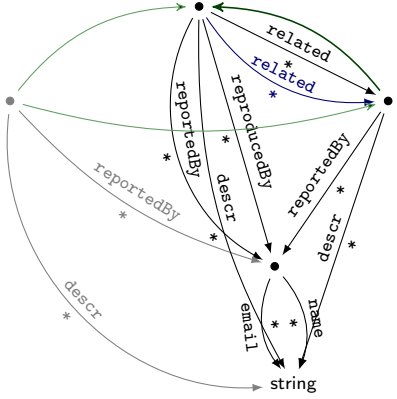
Inference of SingShEx$_0$: Reduction in detail



Removing a node n dominated by m

- remove n and all of its outgoing edges
- redirect its incoming edges to m

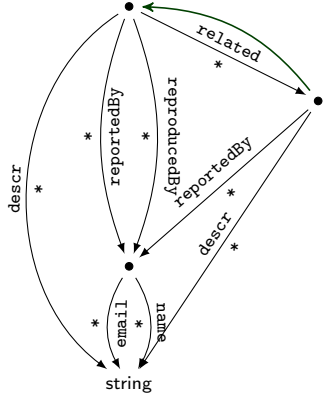
Inference of SingShEx$_0$: Reduction in detail



Removing a node n dominated by m

- remove n and all of its outgoing edges
- redirect its incoming edges to m

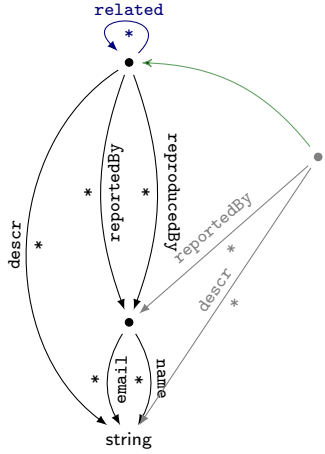
Inference of SingShEx$_0$: Reduction in detail



Removing a node n dominated by m

- remove n and all of its outgoing edges
- redirect its incoming edges to m

Inference of SingShEx$_0$: Reduction in detail

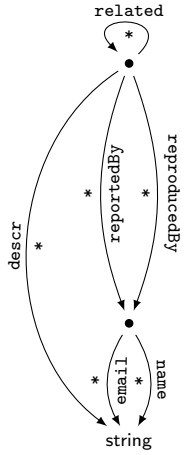


Removing a node n dominated by m

- remove n and all of its outgoing edges
- redirect its incoming edges to m

Inference of SingShEx$_0$: Reduction in detail

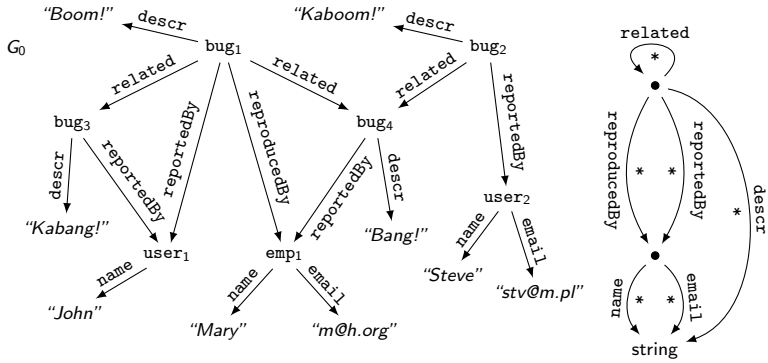
Reduce($G^\star$)



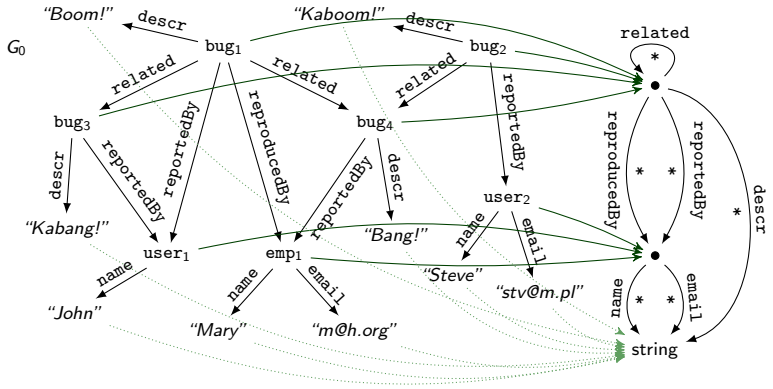
Removing a node n dominated by m

- remove n and all of its outgoing edges
- redirect its incoming edges to m

Inference of SingShEx$_0$: Specialization

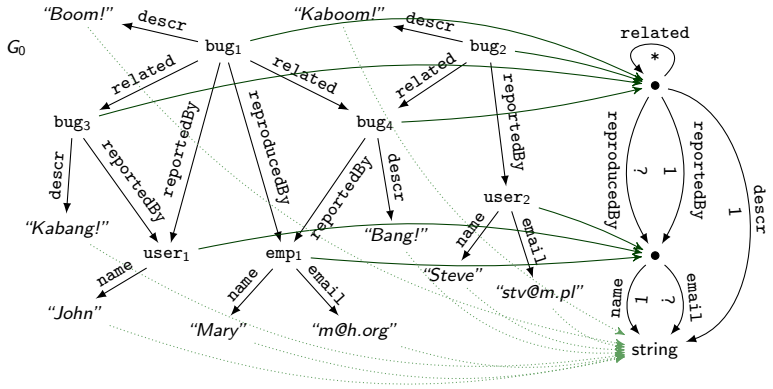


Inference of SingShEx$_0$: Specialization



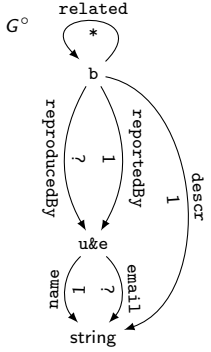
- Construct the embedding of the input graph G into the reduct of G^*

Inference of SingShEx$_0$: Specialization



- Construct the embedding of the input graph G into the reduct of G^*
- Use the embedding to replace $*$'s with fitter multiplicities

Inference of SingShEx$_0$: Specialization



- Construct the embedding of the input graph G into the reduct of G^*
- Use the embedding to replace $*$'s with fitter multiplicities

Singular Shape Expression Schemas

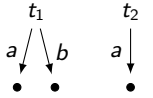
Definition (SingShEx$_0$)

A shape graph H is *singular* if

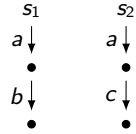
1. $H^\star$ is reduced i.e., there are no two types t_1 and t_2 in $H^\star$ that t_1 can be embedded into t_2,
2. $H^\star$ has no two edges with the same label and the same source and target nodes.

Singularity as a restriction is

- stronger than forbidding any two types t_1 and t_2 such that $t_1 \subseteq t_2$.
- weaker than forbidding any two types with comparable signatures (sets of outgoing edge labels).



Not singular



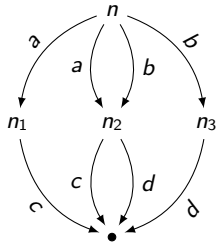
Singular

Deterministic Shape Expression Schemas

Deterministic Shape Expression Schemas

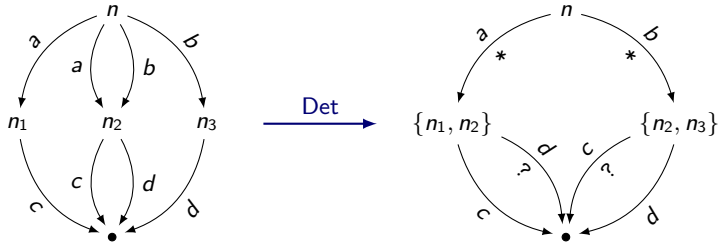
Definition ($\mathrm{DetShEx}_0$)

A shape graph H is *deterministic* if for every label a every node has at most one outgoing edges labeled with a.



Not deterministic

Determinizing Shape Graphs



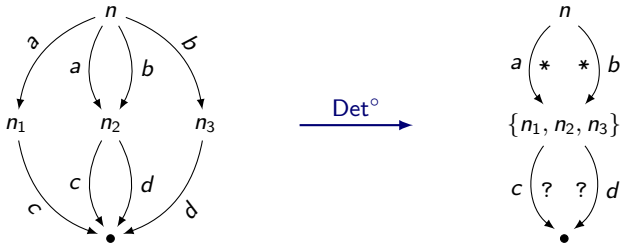
Determinization of a shape graph

- gives the fitting of the input graph
  (the $\subseteq$-minimal DetShEx$_0$ schema that validates the input graph)
- might produce exponential output

Quotient Determinization of Shape Graphs

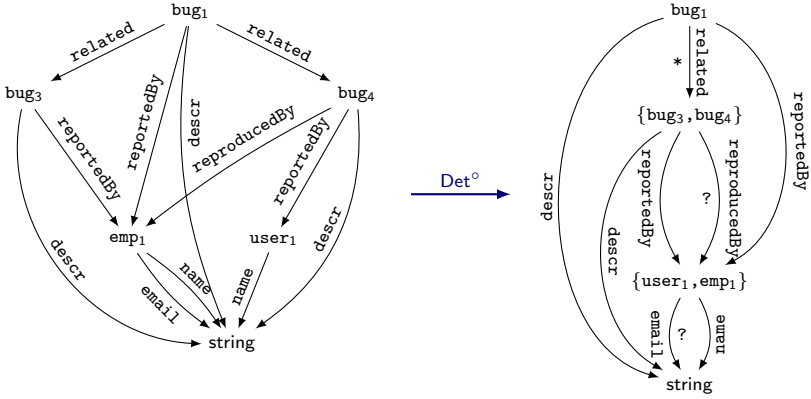
Avoid explosion by refusing to duplicate types

- if n_1 is fused with n_2 and n_2 is fused with n_3, then fuse n_1, n_2, and n_3 together.
- produces linear output



Quotient determinization is an inference algorithm for $\mathrm{DetShEx}_0^-$

Example


bug1
related
related
bug3
bug4
reportedBy
descr
reportedBy
reproducedBy
reportedBy
descr
emp1
user1
descr
name
email
name
string
Det°
bug1
*
related
reportedBy
{bug3, bug4}
reportedBy
?
reproducedBy
descr
descr
{user1, emp1}
email
?
name
string

Conclusions and Future Work

Summary

What is ShEx?

- Automata-like formalism for graphs but unlike any automata we have seen.
- Strong connections to graph bisimulation.

Complexity of Validation

	ShEx_0	ShEx	DetShEx (SORBE)
multi-type	PTIME	NP-complete	PTIME
single-type	NP-complete		

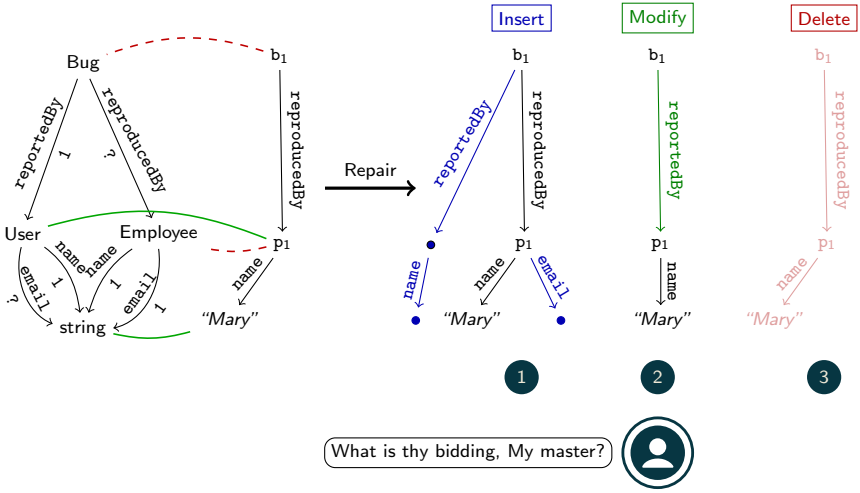
Complexity of Containment

ShEx	DetShEx(SORBE)	ShEx_0	DetShEx_0	DetShEx_0^-
coNEXP-h and co2EXP^{NP}	co2EXP	EXP-c	coNP-h	PTIME

Inference and Fitting

	ShEx_0	SingShEx_0	DetShEx_0	Typed graph
Fitting	trivial	undefined (?)	exponential	NP-hard
Inference	unfeasible	$\text{SingShEx}_0(1, *)$	DetShEx_0^-	ShEx_0 (?)

Future work: Repairing RDF



Questions

Appendix

Formal Definition of Grammatical Inference of Shape Expression Schemas

Definition

A class of shape graphs $\mathcal{C}$ is *learnable in polynomial time and data from* a class of graphs $\mathcal{G}$ iff there exists a polynomial inference algorithm A such that the following two conditions are satisfied:

Soundness For every input graph $G \in \mathcal{G}$ the inference algorithm returns a graph schema $A(G) = H$ such that $H \in \mathcal{C}$ and $G \in L(H)$.

Completeness For every graph schema $H \in \mathcal{C}$ there exists a polynomially-sized *characteristic* graph $G_H \in L(H)$ such that for any G that extends G_H consistently with H we have $A(G') \equiv H$.

When does G extend G'?

3 alternative definitions

1. G is a disjoint union of G' and some G''
2. G is obtained from G' by adding new nodes and new edges
3. as 2. but no node of G may loose a type as a result of adding outgoing edges to it.