Complexity of RDF Validation with Shape Expression Schemas

Sławek Staworko[1,2,3]
(joint work with Iovka Boneva[1,2], Jose E. Labra Gayo[4], Samuel Hym[2], Eric G. Prud'hommeaux[5], and Harold Solbrig[6])

[1]LINKS, INRIA & CNRS, Lille, France

[2]CRIStAL, University of Lille & CNRS, France

[3]University of Edinburgh, UK

[4]University of Oviedo, Spain

[5]W3C, Stata Center, MIT

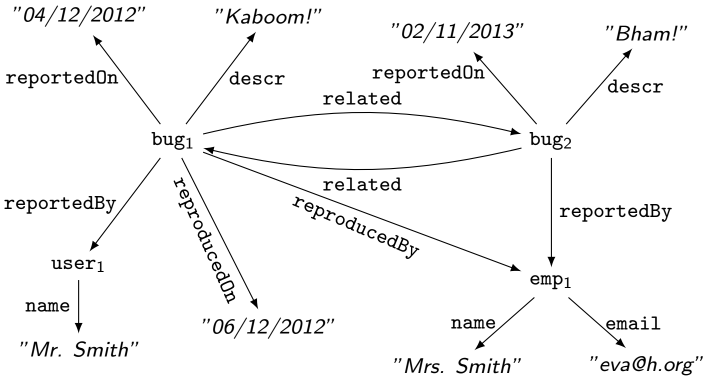
[6]Mayo Clinic College of Medicine, Rochester, MN, USA

Oxford University

May 14, 2015

Background: RDF Graphs

RDF Graph = set of triples ⟨*subject predicate object*⟩



Originally, introduced as schema-less data format.

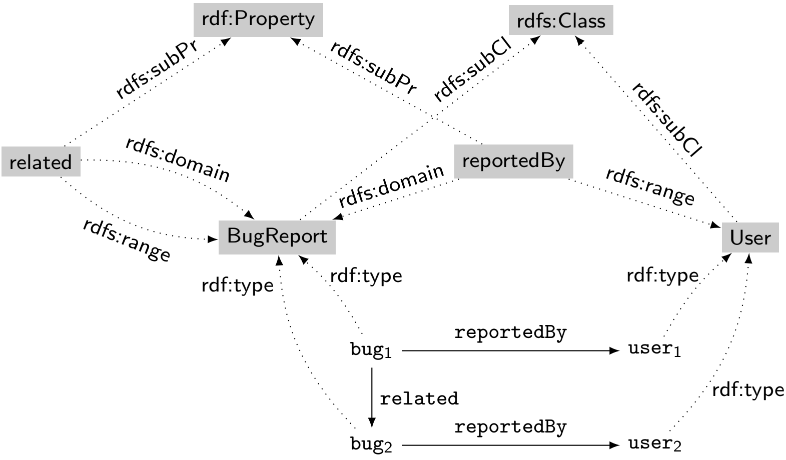
Overview

1. Existing schema formalisms
2. Shape expression schemas and their two semantics
3. Intractability of single-type semantics
4. Complexity of multi-type semantics
5. Determinism, single-occurrence, . . .

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



Existing schema formalisms for RDF (cont..)

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 entail:

Verification : the typing is given (with rdfs:type) and its correctness is to be verified; this variant is adopted by all existing formalisms.

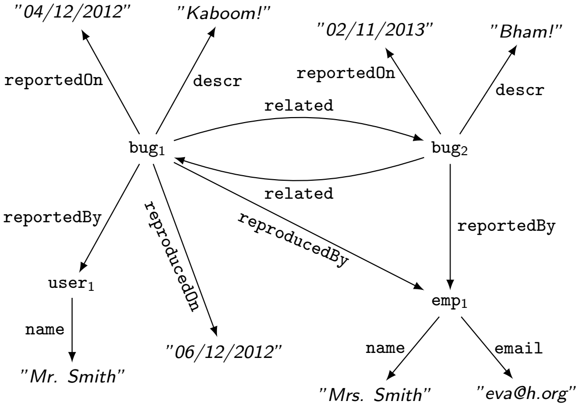
Model checking : not typing is given and the goal is to construct a valid typing; this is more general problem that we address in this work

Shape expressions schemas for RDF

```
<BugReport> {
  descr xsd:string,
  reportedBy @<User>,
  reportedOn xsd:dateTime,
  (reproducedBy @<Employee>,
   reproducedOn xsd:dateTime)?,
  related @<BugReport>*
}

<User> {
  name xsd:string,
  email xsd:string?
}

<Employee> {
  (name xsd:string |
   (first-name xsd:string,
    last-name xsd:string)),
  email xsd:string,
}
```



Basics: RDF graphs

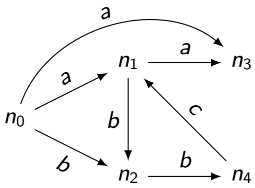
We assume a fixed finite set Σ of edge labels.

We model RDF with edge-labeled graphs: $G = (V, E)$, where $E \subseteq V \times \Sigma \times V$.

We shall constraint the structure of RDF Graph by imposing type constriants on the outbound neighborhood of a node:

$$out\text{-}lab\text{-}node_G(n) = \{(a, m) \in \Sigma \times V \mid (n, a, m) \in E\}$$

G_0:



$$out\text{-}lab\text{-}node_{G_0}(n_0) = \{(a, n_1), (b, n_2), (a, n_3)\}$$

What is the collection of labels of outgoing edges of n_0?

Basics: Bags of symbols (unordered words)

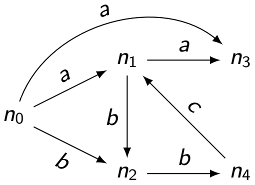
Bag (multiset) is a function mapping a symbol to the number of its occurrences.

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

The collection of outgoing labels is a bag:

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

G_0:



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

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 $\parallel$)

$$E ::= \epsilon \mid a \mid E^* \mid (E \text{``}|\text{''} E) \mid (E \text{``}\|\text{''} E)$$

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

Examples

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

RBEs are equivalent to

1. Presburger arythmetic,
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 a simple and well-behaved subclass of RBEs

RBE_0

- RBEs using only symbols with multiplicities $\{0, 1, *, +, ?\}$ and $\parallel$ operator only
- can be canonized $a \parallel a^? \equiv a^{[1,2]}$, $b^+ \parallel b^+ \equiv b^{[2,\infty]}$, etc.
- the canonical form is $a^{[n,n']} \parallel b^{[m,m']} \parallel \ldots$
- 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]

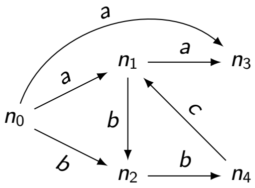
Shape expression schemas

A Shape Expression Schema is a tuple $S = (\Sigma, \Gamma, \delta)$, where

- Γ is a finite set of types,
- δ maps types to type definitions (RBEs over $\Sigma \times \Gamma$)
  $a :: t$ stands for (a, t)

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

G_0:



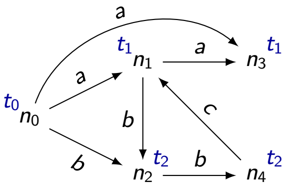
Shape expression schemas

A Shape Expression Schema is a tuple $S = (\Sigma, \Gamma, \delta)$, where

- Γ is a finite set of types,
- δ maps types to type definitions (RBEs over $\Sigma \times \Gamma$)
  $a :: t$ stands for (a, t)

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

G_0:



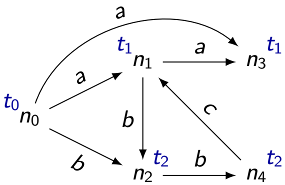
Shape expression schemas

A Shape Expression Schema is a tuple $S = (\Sigma, \Gamma, \delta)$, where

- Γ is a finite set of types,
- δ maps types to type definitions (RBEs over $\Sigma \times \Gamma$)
  $a :: t$ stands for (a, t)

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

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.,

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

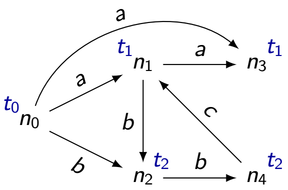
Shape expression schemas

A Shape Expression Schema is a tuple $S = (\Sigma, \Gamma, \delta)$, where

- Γ is a finite set of types,
- δ maps types to type definitions (RBEs over $\Sigma \times \Gamma$)
  $a :: t$ stands for (a, t)

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

G_0:



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

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

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

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

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 for RBE_0*).*

Reduction from graph 3-colorability:

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

Multi-type semantics of shape expressions

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

$$
\begin{aligned}
S_1\colon\quad & t_0 \rightarrow a :: t_1 \\
& t_1 \rightarrow b :: t_2 \parallel c :: t_3 \\
& t_2 \rightarrow (b :: t_2)^{?} \parallel c :: t_3 \\
& t_3 \rightarrow \epsilon
\end{aligned}
$$

$$
G_1\colon \quad
\begin{tikzcd}
n_0 \arrow[r, "a"] & n_1 \arrow[loop above, "b"] \arrow[r, "c"] & n_2
\end{tikzcd}
$$

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.

$$
\begin{aligned}
S_1\colon\quad & t_0 \to a :: t_1 \\
& t_1 \to b :: t_2 \parallel c :: t_3 \\
& t_2 \to (b :: t_2)^{?} \parallel c :: t_3 \\
& t_3 \to \epsilon
\end{aligned}
$$

$$
G_1\colon\quad
\begin{tikzcd}
n_0 \arrow[r, "a"] & n_1 \arrow[r, "c"] \arrow[loop above, "b"] & n_2 \\
t_0 & &
\end{tikzcd}
$$

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.

$$
\begin{aligned}
S_1\colon\quad & t_0 \to a :: t_1 \\
& t_1 \to b :: t_2 \parallel c :: t_3 \\
& t_2 \to (b :: t_2)^? \parallel c :: t_3 \\
& t_3 \to \epsilon
\end{aligned}
$$

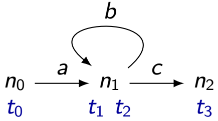
$$
G_1\colon\quad
\begin{tikzcd}
n_0 \arrow[r, "a"] & n_1 \arrow[loop above, "b"] \arrow[r, "c"] & n_2 \\[-20pt]
t_0 & t_1 &
\end{tikzcd}
$$

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.

$$
\begin{aligned}
S_1:\quad & t_0 \to a :: t_1 \\
& t_1 \to b :: t_2 \parallel c :: t_3 \\
& t_2 \to (b :: t_2)^{?} \parallel c :: t_3 \\
& t_3 \to \epsilon
\end{aligned}
$$

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.

$$
\begin{aligned}
S_1\colon\quad & t_0 \to a :: t_1 \\
& t_1 \to b :: t_2 \parallel c :: t_3 \\
& t_2 \to (b :: t_2)^{?} \parallel c :: t_3 \\
& t_3 \to \epsilon
\end{aligned}
$$

$$
G_1\colon \quad
\begin{tikzcd}
n_0 \arrow[r, "a"] & n_1 \arrow[r, "c"] \arrow[loop above, "b"] & n_2 \\
t_0 & t_1 \;\; t_2 & t_3
\end{tikzcd}
$$

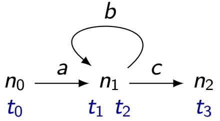
$$
\lambda_1(n_0) = \{t_0\}, \qquad \lambda_1(n_1) = \{t_1, t_2\}, \qquad \lambda_1(n_2) = \{t_3\}.
$$

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.

$$
\begin{aligned}
S_1\colon\ & t_0 \to a :: t_1 \\
& t_1 \to b :: t_2 \parallel c :: t_3 \\
& t_2 \to (b :: t_2)^? \parallel c :: t_3 \\
& t_3 \to \epsilon
\end{aligned}
$$

G_1:



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

Talk declarative to me

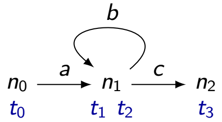
λ is valid if every node satisfies every of its associated types.

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.

$$
\begin{aligned}
S_1\colon\quad & t_0 \to a :: t_1\\
& t_1 \to b :: t_2 \parallel c :: t_3\\
& t_2 \to (b :: t_2)^{?} \parallel c :: t_3\\
& t_3 \to \epsilon
\end{aligned}
$$

G_1:



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

Talk declarative to me

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

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

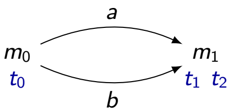
- we inspect the outbound neighborhood $\textit{out-lab-node}_G(n) = \{(a, m) \mid (n, a, m) \in E\}$

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.

$$S_2:\quad \begin{aligned} t_0 &\to a :: t_1 \parallel b :: t_2 \\ t_1 &\to (c :: t_1)^* \\ t_2 &\to (d :: t_2)? \end{aligned}$$

G_2:



Talk declarative to me

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

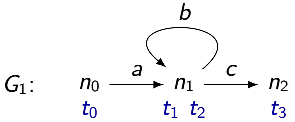
When defining that a node n satisfies a type t...

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

$$
\begin{aligned}
S_1:\ & t_0 \to a :: t_1 \\
& t_1 \to b :: t_2 \parallel c :: t_3 \\
& t_2 \to (b :: t_2)^{?} \parallel c :: t_3 \\
& t_3 \to \epsilon
\end{aligned}
$$



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

Talk declarative to me

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

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

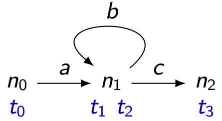
- we inspect the outbound neighborhood $\mathit{out\text{-}lab\text{-}node}_G(n) = \{(a, m) \mid (n, a, m) \in E\}$
- a node m may assume any of its assigned types $\lambda(m)$, one per edge incoming from n
- $(n, a, m) \in E$ yields the choice $|_{t \in \lambda(m)}\ a :: t$

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.

$$
\begin{aligned}
S_1\colon\quad & t_0 \to a :: t_1 \\
& t_1 \to b :: t_2 \parallel c :: t_3 \\
& t_2 \to (b :: t_2)^? \parallel c :: t_3 \\
& t_3 \to \epsilon
\end{aligned}
$$

G_1:



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

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

Talk declarative to me

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 $\textit{out-lab-node}_G(n) = \{(a, m) \mid (n, a, m) \in E\}$
- 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 $|_{t \in \lambda(m)}\ a :: t$
- $OutType(n, \lambda) = \parallel_{(n,a,m) \in E} (|_{t \in \lambda(m)}\ a :: t)$

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.

$$
S_1\colon \quad
\begin{aligned}
& t_0 \to a :: t_1 \\
& t_1 \to b :: t_2 \parallel c :: t_3 \\
& t_2 \to (b :: t_2)^{?} \parallel c :: t_3 \\
& t_3 \to \epsilon
\end{aligned}
$$

$$
G_1\colon \quad
\begin{tikzcd}
n_0 \arrow[r, "a"] & n_1 \arrow[r, "c"] \arrow[loop above, "b"] & n_2 \\
t_0 & t_1 \; t_2 & t_3
\end{tikzcd}
$$

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

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

Talk declarative to me

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

$$\boxed{\begin{array}{c} n \text{ satisfies } t \text{ w.r.t. } \lambda \text{ if } OutType(n, \lambda) \cap \delta(t) \neq \emptyset, \\ \text{where } OutType(n, \lambda) = \parallel_{(n,a,m) \in E} (|_{t \in \lambda(m)}\ a :: t) \end{array}}$$

Refinement algorithm

$\boxed{\text{The set of all valid multi-type typings of } G \text{ w.r.t. } S \text{ is a semi-lattice.}}$

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)

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

The essential decision problem for a class $\mathcal{C}$ of RBEs used in the 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 schemas using 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.

$a :: t_1 \parallel b :: t_2^* \parallel a :: t_1 \parallel c :: t_2$	$(a :: t_1 \parallel b :: t_2) \mid (a :: t_3 \parallel c :: t_4)$	$a :: t_1 \parallel b :: t_2^* \parallel a :: t_3$
deterministic	not deterministic	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.

$a :: t_1 \parallel b :: t_2^* \parallel a :: t_1 \parallel c :: t_2$	$(a :: t_1 \parallel b :: t_2) \mid (a :: t_3 \parallel c :: t_4)$	$a :: t_1 \parallel b :: t_2^* \parallel a :: t_3$
deterministic	not deterministic	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
- $\mathit{OutType}(n, \lambda) = \parallel_{(n,a,m)\in E}(|_{t\in\lambda(m)}\ a :: t)$ becomes $\parallel_{(n,a,m)\in E}(a :: t_a)$
- $\parallel_{(n,a,m)\in E}(a :: t_a)$ defines a singleton $\{w\}$ with $w = \{\!| a :: t_a \mid (n, a, m) |\!\}$
- $\mathit{OutType}(n, \lambda) \cap \delta(t) \neq \emptyset \equiv w \in \delta(t)$.

Going further: unambiguity

Unambiguity of shape expressions

Given the type, the context (all labels on outgoing edges), and the label of an outgoing edge, the expression specifies at most one type for the target node.

$a :: t_1 \parallel b :: t_2^* \parallel a :: t_1 \parallel c :: t_2$	$(a :: t_1 \parallel b :: t_2) \mid (a :: t_3 \parallel c :: t_4)$	$a :: t_1 \parallel b :: t_2^* \parallel a :: t_3$
deterministic and unambiguous	not deterministic but unambiguous	not deterministic and not unambiguous

Theorem

Testing unambiguity is coNP-complete :(

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 :)

$a :: t_1 \parallel b :: t_2^* \parallel a :: t_1$	$(a :: t_1 \parallel b :: t_2) \mid (a :: t_3 \parallel c :: t_4)$	$(a :: t_1 \parallel b :: t_2)^* \parallel c :: t_3$
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. :)*

Validation with pretyping

Pretyping is a typing $\lambda_{\_} : V \to 2^{\Gamma}$ that need not be valid (given on input)

Universal type $t_{\top}$ is satisfied by all nodes i.e., $\delta(t_{\top}) = (\Sigma \times \Gamma)^*$

Goal: Find an extension of $\lambda_{\_}$ that is a valid multi-type typing.

Lemma

For deterministic shape expression schemas with universal type, if $\lambda_{\_}$ admits a valid extension, then it admits a unique minimal valid extension.

Theorem

The minimal valid extension of a pretyping can be constructed in polynomial time for deterministic shape expression schemas that use SORBE

Flooding algorithm

Algorithm 1 MinValidExt$(S, G, \lambda_{\text{-}})$

Input: $S = (\Sigma, \Gamma, \delta)$ a deterministic ShEx,
$G = (V, E)$,
$\lambda_{\text{-}} \subseteq V \times \Gamma$ a pre-typing;

Output: $\lambda \subseteq V \times \Gamma$ the minimal valid extension of $\lambda_{\text{-}}$.

let $F := \lambda_{\text{-}}$
let $\lambda := \emptyset$
while $F \neq \emptyset$ **do**
  choose $(n, t) \in F$ and remove it from F
  let $\mathit{out\text{-}lab\text{-}type}_G^{\delta}(n, t) := \{\!|(a, t_a^{\delta(t)}) \mid (a, m) \in \mathit{out\text{-}lab\text{-}node}_G(n)|\!\}$
  if $\mathit{out\text{-}lab\text{-}type}_G^{\delta}(n, t) \notin \delta(t)$ **then**
    fail
  $\lambda := \lambda \cup \{(n, t)\}$
  for $(a, m) \in \mathit{out\text{-}lab\text{-}node}_G(n)$ **do**
    if $t_a^{\delta(t)} \neq t_\top$ **and** $(m, t_a^{\delta(t)}) \notin \lambda$ **then**
      $F := F \cup \{(m, t_a^{\delta(t)})\}$
return λ

Summary

1. Formalization of shape expression schemas
2. Two semantics (single- and multi-type)
3. Identification of complexity bottlenecks:
   - Intersection with RBE_1 for arbitrary shape expressions
   - Membership for deterministic shape expressions
4. Initial complexity analysis

	RBE_0	RBE	SORBE	SORBE det.	SORBE det. $+ \lambda_{\_} + t_{\top}$
multi-type	PTIME	NP-complete		PTIME	
single-type	NP-complete				PTIME

5. Initial analysis of expressive power
   - automata-like formalism
   - incomparable to FO and MSO (unless we forbid $*$ over expressions)
   - 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

Future work

1. Continuing work (W3C)
2. Popularization effort
3. Identifying tractable and practical subclasses of RBE
4. Hybrid flooding/refinement algorithm
5. Inference of shape expression schemas