

Segunda prueba Trabajo de investigación

Jose Emilio Labra Gayo



Universidad de Oviedo

Creación de subconjuntos de grafos de conocimiento mediante Shape Expressions

Jose Emilio Labra Gayo



Razones para elección del tema

Continúa investigación sobre *Shape Expressions* y sus aplicaciones

Resolver problema práctico, propuesto por la comunidad

Participación en biohackathones con esa temática

- Virtual Biohackathon 2020

- SWAT4(HC)LS Hackathon 2021

Adopción de *Shape Expressions* por Wikidata

Contrato "*Robustifying Scholia*", Univ. Virginia, *Sloan Foundation*

Interés de Andra Waagmeester y proyecto GeneWiki

Parte de entregable proyecto ANGLIRU

Avance en la disciplina: Mejorar/facilitar consumo de datos semánticos

Principales resultados del artículo

Definición formal de grafos RDF, *property graphs* y *wikibase graphs*

Extensión de ShEx para *property graphs* (PShEx)

Extensión de ShEx para *wikibase graphs* (WShEx)

Definición de 5 técnicas de generación de subconjuntos

- Entity-matching

- Simple matching

- ShEx-based matching

- Shex+Slurp

- ShEx+Pregel

Definición e implementación algoritmo validación a gran escala sobre Spark GraphX

Formato del trabajo

Artículo científico en inglés

La convocatoria no indica requisitos de idioma

El inglés permite su posterior presentación en revista/congreso

Publicado en abierto como preprint en Arxiv

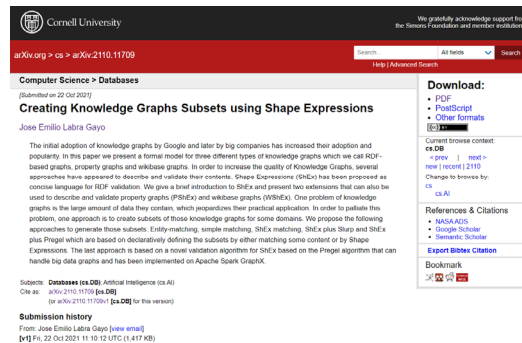
<https://arxiv.org/abs/2110.11709>

Objetivos a corto plazo:

Optimizar implementaciones

Terminar experimentos

Enviar a publicación



Creating Knowledge Graphs subsets using Shape Expressions

Jose Emilio Labra Gayo



Motivation and main idea (1)

Since its proposal in 2012, KGs are very successful

Different kinds of KGs

- Free/open and collaborative like Wikidata
- Enterprise/proprietary and owned by companies: Google, Amazon, eBay, Facebook, Microsoft,...

Size of their contents is continually growing

Problem: KGs victims of their own success?

Example: size of Wikidata dumps almost doubling

- 30GB (2014 - uncompressed)
- 1.256GB (2021 - uncompressed)

Difficult for produce/consume/handle KGs



Idea 1: create subset of KGs for some domain



Motivation and main idea (2)

Shape Expressions: language to describe and validate RDF

- Concise and human-friendly
- Automatically procesable

Several tools already exist around ShEx

Target audience: domain data experts

Already adopted by Wikidata to validate RDF serialization of Wikibase entities

```
<Researcher> {
  :name      xsd:string ;
  :birthDate xsd:date ? ;
  :birthPlace @<Place> ? ;
  :knows     @<Researcher> *
}
<Place> {
  :name      xsd:string ;
  :country   @<Country>
}
<Country> {
  :name      xsd:string ;
}
```

Example of a ShEx schema

Idea 2: use Shape expressions to define those subsets



Structure of the presentation

- 3 main types of Knowledge graphs
- {

RDF graphs

Property graphs

Wikibase graphs

}
- ShEx for those types of Knowledge Graphs
- {

ShEx

PSHEx

WShEx

}
- Review and propose approaches to generate subsets:
- {

Entity matching

Simple matching

ShEx-based matching

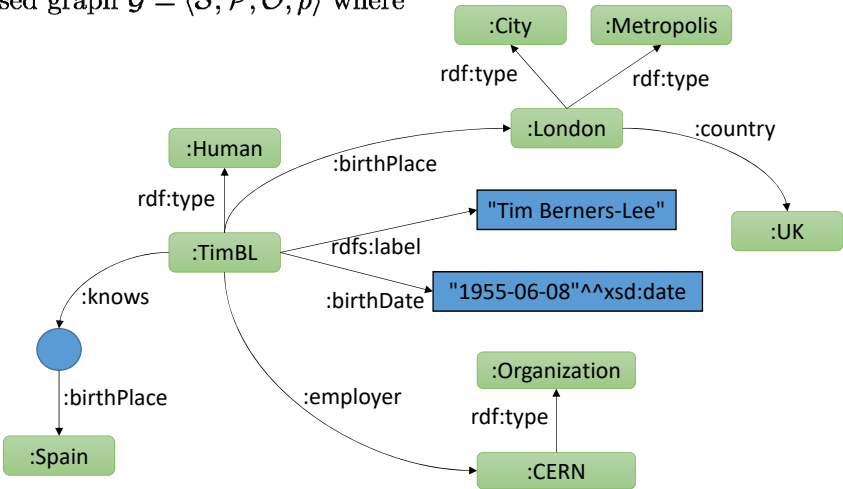
ShEx+Slurp

ShEx+Pregel

}

RDF graphs

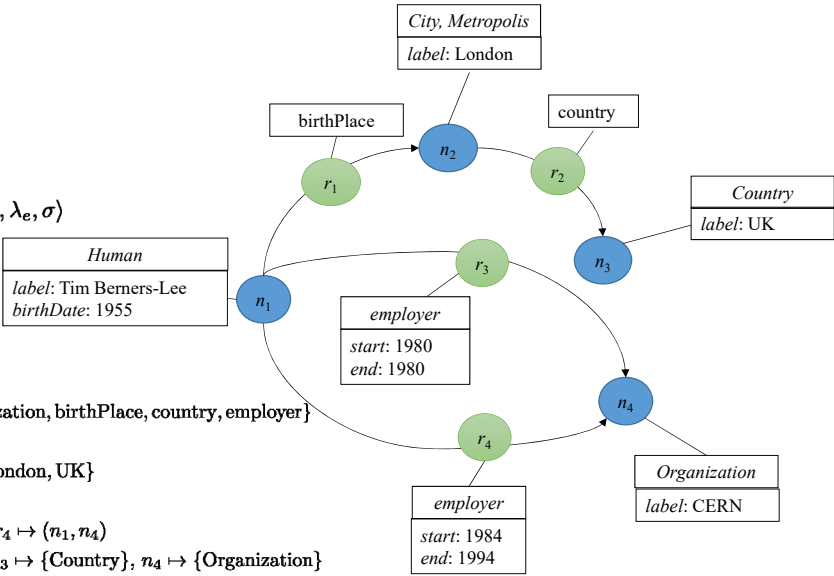
Given a set of IRIs \mathcal{I} , a set of blank nodes \mathcal{B} and a set of literals \mathcal{Lit}
an *RDF graph* is a triple based graph $\mathcal{G} = \langle \mathcal{S}, \mathcal{P}, \mathcal{O}, \rho \rangle$ where
 $\mathcal{S} = \mathcal{I} \cup \mathcal{B}$,
 $\mathcal{P} = \mathcal{I}$,
 $\mathcal{O} = \mathcal{I} \cup \mathcal{B} \cup \mathcal{Lit}$
 $\rho \subseteq \mathcal{S} \times \mathcal{P} \times \mathcal{O}$



Property graphs

Given a set of types \mathcal{T} ,
a set of properties \mathcal{P} ,
and a set of values \mathcal{V} ,
a *property graph* \mathcal{G} is a tuple $\langle \mathcal{N}, \mathcal{E}, \rho, \lambda_n, \lambda_e, \sigma \rangle$
where $\mathcal{N} \cap \mathcal{E} = \emptyset$,
 $\rho : \mathcal{E} \mapsto \mathcal{N} \times \mathcal{N}$ is a total function
 $\lambda_n : \mathcal{N} \mapsto FinSet(\mathcal{T})$
 $\lambda_e : \mathcal{E} \mapsto \mathcal{T}$
 $\sigma : \mathcal{N} \cup \mathcal{E} \times \mathcal{P} \mapsto FinSet(\mathcal{V})$

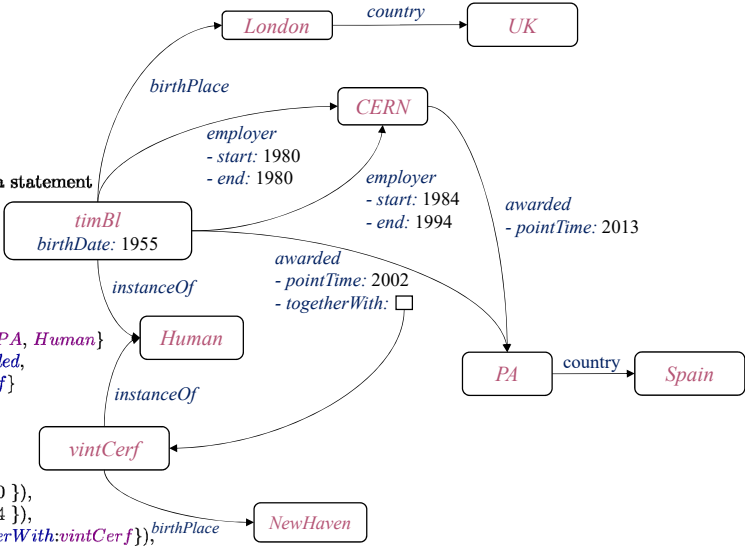
$\mathcal{T} = \{Human, City, Metropolis, Country, Organization, birthPlace, country, employer\}$
 $\mathcal{P} = \{label, birthDate, start, end\}$
 $\mathcal{V} = \{Tim\ Berners\text{-}Lee, 1955, 1980, 1984, 1994, London, UK\}$
 $\mathcal{N} = \{n_1, n_2, n_3, n_4\} \quad \mathcal{E} = \{r_1, r_2, r_3, r_4\}$
 $\rho = r_1 \mapsto (n_1, n_2), r_2 \mapsto (n_2, n_3), r_3 \mapsto (n_1, n_4), r_4 \mapsto (n_1, n_4)$
 $\lambda_n = n_1 \mapsto \{Human\}, n_2 \mapsto \{City, Metropolis\}, n_3 \mapsto \{Country\}, n_4 \mapsto \{Organization\}$
 $\lambda_e = r_1 \mapsto birthPlace, r_2 \mapsto country, r_3 \mapsto employer, r_4 \mapsto employer$
 $\sigma = (n_1, label) \mapsto Tim\ Berners\text{-}Lee, (n_1, birthDate) \mapsto 1955$
 $(n_2, label) \mapsto London, (n_3, label) \mapsto UK, (n_4, label) \mapsto CERN$
 $(r_3, start) \mapsto 1980, (r_3, end) \mapsto 1980, (r_4, start) \mapsto 1984, (r_4, end) \mapsto 1994$



Wikibase graphs

Given a mutually disjoint set of items \mathcal{Q} ,
a set of properties \mathcal{P} and
a set of data values \mathcal{D} ,
a *Wikibase graph* is a tuple $\langle \mathcal{Q}, \mathcal{P}, \mathcal{D}, \rho \rangle$ such that
 $\rho \subseteq \mathcal{E} \times \mathcal{P} \times \mathcal{V} \times FinSet(\mathcal{P} \times \mathcal{V})$ where
 $\mathcal{E} = \mathcal{Q} \cup \mathcal{P}$ is the set of entities which can be subjects of a statement
 $\mathcal{V} = \mathcal{E} \cup \mathcal{D}$ is the set of possible values of a property.

$\mathcal{Q} = \{timBl, vintCerf, London, CERN, UK, Spain, PA, Human\}$
 $\mathcal{P} = \{birthDate, birthPlace, country, employer, awarded, start, end, pointTime, togetherWith, instanceOf\}$
 $\mathcal{D} = \{1984, 1994, 1980, 1955\}$
 $\rho = \{(timBl, instanceOf, Human, \{\}), (timBl, birthDate, 1955, \{\}), (timBl, birthPlace, London, \{\}), (timBl, employer, CERN, \{start: 1980, end: 1980\}), (timBl, employer, CERN, \{start: 1984, end: 1994\}), (timBl, awarded, PA, \{pointTime: 2002, togetherWith: vintCerf\}), (London, country, UK, \{\}), (vintCerf, instanceOf, Human, \{\}), (vintCerf, birthPlace, NewHaven, \{\}), (CERN, awarded, PA, \{pointTime: 2013\}), (PA, country, Spain, \{\})\}$



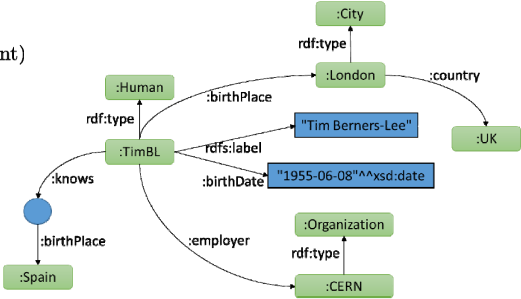
ShEx for RDF

A ShEx Schema is a tuple $\langle \mathcal{L}, \delta \rangle$ where

\mathcal{L} set of shape labels

$\delta : \mathcal{L} \rightarrow se$

$se ::=$	$cond$	Basic boolean condition on nodes (node constraint)
	s	Shape
	$se_1 \text{ AND } se_2$	Conjunction
	$se_1 \text{ OR } se_2$	Disjunction
	$\text{NOT } se$	Negation
	$@l$	Shape label reference for $l \in \mathcal{L}$
$s ::=$	$\text{CLOSED } \{te\}$	Closed shape
	$\{te\}$	Open shape
$te ::=$	$te_1; te_2$	Each of te_1 and te_2
	$te_1 \mid te_2$	Some of te_1 or te_2
	te^*	Zero or more te
	ϵ	Empty triple expression
	$\neg \xrightarrow{p} @l$	Triple constraint with predicate p



\mathcal{L}	$= \{ \text{Person, Place, Country, Organization, Date} \}$
$\delta(\text{Person})$	$= \{ \neg \xrightarrow{\text{birthDate}} @\text{Date}; \neg \xrightarrow{\text{birthPlace}} @\text{Place}; \neg \xrightarrow{\text{employer}} @\text{Organization}^* \}$
$\delta(\text{Place})$	$= \{ \neg \xrightarrow{\text{country}} @\text{Country} \}$
$\delta(\text{Country})$	$= \{ \}$
$\delta(\text{Organization})$	$= \{ \}$
$\delta(\text{Date})$	$= \text{xsd:Date}$

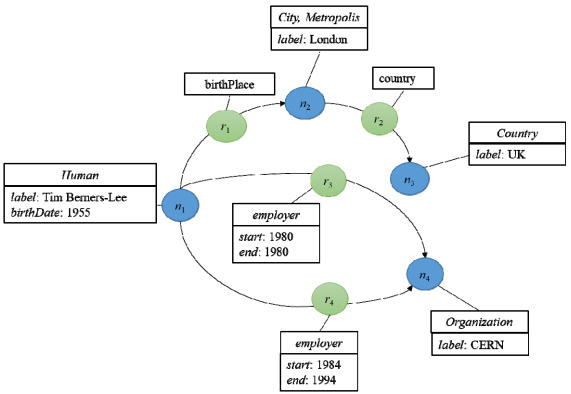
PShEx for property graphs

A PShEx Schema is a tuple $\langle \mathcal{L}, \delta \rangle$ where

\mathcal{L} set of shape labels

$\delta : \mathcal{L} \rightarrow se$

$se ::=$	$cond_{t_s}$	Basic boolean condition on set of types $t_s \subseteq \mathcal{T}$
	s	Shape
	$se_1 \text{ AND } se_2$	Conjunction
	$se_1 \text{ OR } se_2$	Disjunction
	$\text{NOT } se$	Negation
	$@l$	Shape label reference for $l \in \mathcal{L}$
	qs	Qualifiers of that node
$s ::=$	$\text{CLOSED } \{te\}$	Closed shape
	$\{te\}$	Open shape
$te ::=$	$te_1; te_2$	Each of te_1 and te_2
	$te_1 \mid te_2$	Some of te_1 or te_2
	te^*	Zero or more te
	$\neg \xrightarrow{p} @l \text{ } qs$	Triple constraint with property type p whose nodes satisfy the shape l and qualifiers qs
$qs ::=$	$[ps]$	Open qualifier specifiers ps
	$[ps]$	Closed qualifier specifiers ps
$ps ::=$	ps_1, ps_2	Each of ps_1 and ps_2
	$ps_1 \mid ps_2$	OneOf of ps_1 or ps_2
	ps^*	zero of more ps
	$p : cond_v$	Property p with value conforming to $cond_v$
		$cond_{v_s}$ is a boolean condition on sets of values $v_s \subseteq \mathcal{V}$

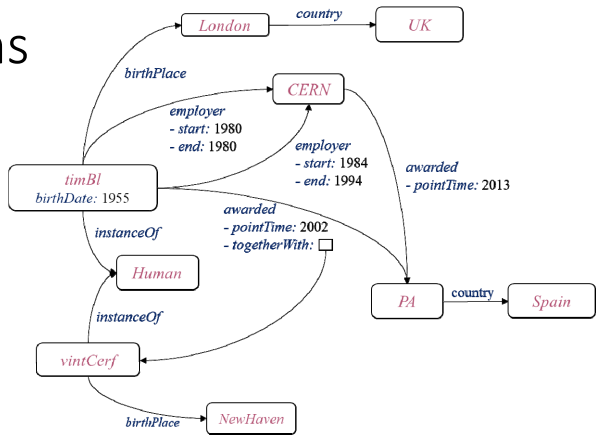


\mathcal{L}	$= \{ \text{Person, Place, Country, Org} \}$
$\delta(\text{Person})$	$= \text{hasType}_{\text{Human}} \text{ AND } [label : String, birthDate : Date] \text{ AND } \{ \neg \xrightarrow{\text{birthPlace}} @\text{Place}; \neg \xrightarrow{\text{employer}} @\text{Org } [start : Date, end : Date]^* \}$
$\delta(\text{Place})$	$= [label : String] \text{ AND } \{ \neg \xrightarrow{\text{country}} @\text{Country} \}$
$\delta(\text{Country})$	$= \text{hasType}_{\text{Country}} \text{ AND } [label : String] \{ \}$
$\delta(\text{Org})$	$= \text{hasType}_{\text{Organization}} \text{ AND } [label : String] \{ \}$

WShEx for Wikibase graphs

A WShEx Schema is a tuple $\langle \mathcal{L}, \delta \rangle$ where
 \mathcal{L} set of shape labels
 $\delta : \mathcal{L} \rightarrow se$

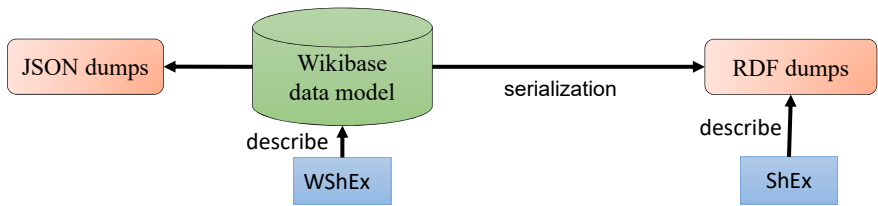
$se ::=$	$cond$	Basic boolean condition on nodes (node constraint)
	s	Shape
	$se_1 \text{ AND } se_2$	Conjunction
	$se_1 \text{ OR } se_2$	Disjunction
	$\text{NOT } se$	Negation
	$@l$	Shape label reference for $l \in \mathcal{L}$
$s ::=$	$\text{CLOSED } s'$	Closed shape
	s'	Open shape
	$\{ te \}$	Shape definition
$te ::=$	$te_1; te_2$	Each of te_1 and te_2
	$te_1 te_2$	Some of te_1 or te_2
	te^*	Zero or more te
	$\neg \overrightarrow{p} @l qs$	Triple constraint with predicate p value conforming to l and qualifier specifier qs
$qs ::=$	ϵ	Empty triple expression
	$[ps]$	Open property specifier
	$[ps]$	Closed property specifier
$ps ::=$	ps, ps	EachOf property specifiers
	$ps ps$	OneOf property specifiers
	ps^*	zero of more property specifiers
	ϵ	Empty property specifier
	$p:@l$	Property p with value conforming to shape l

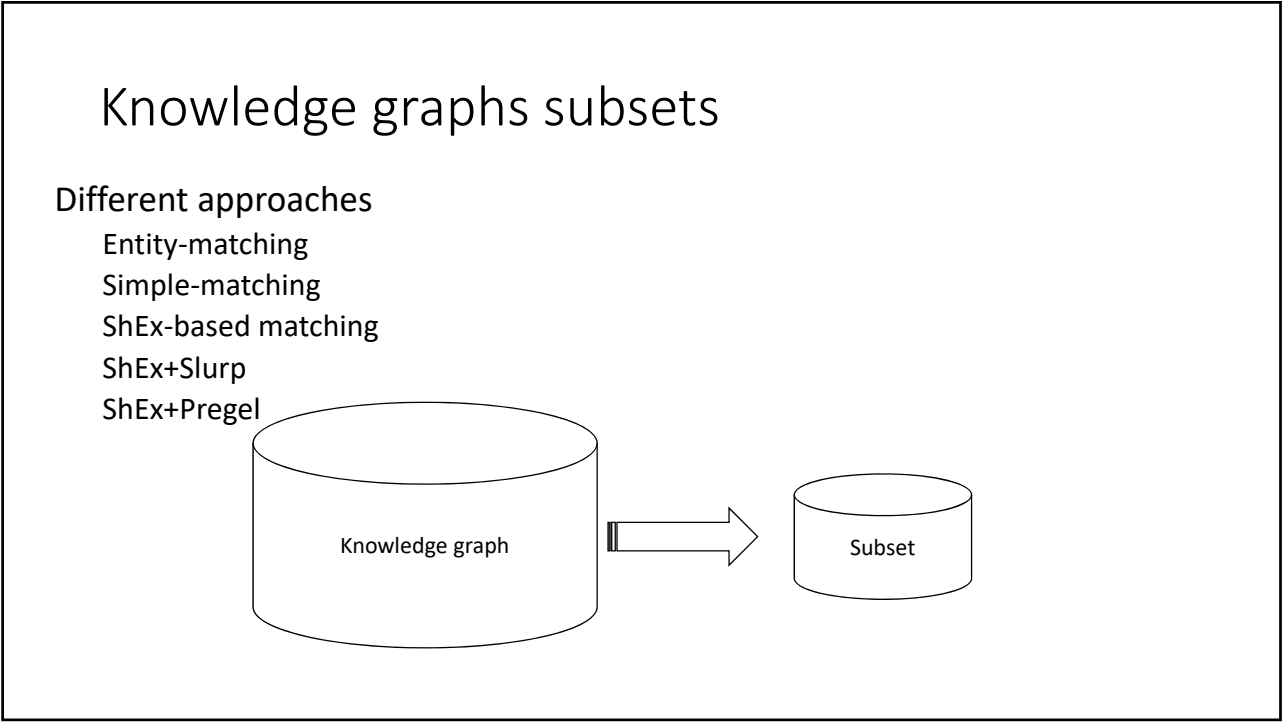
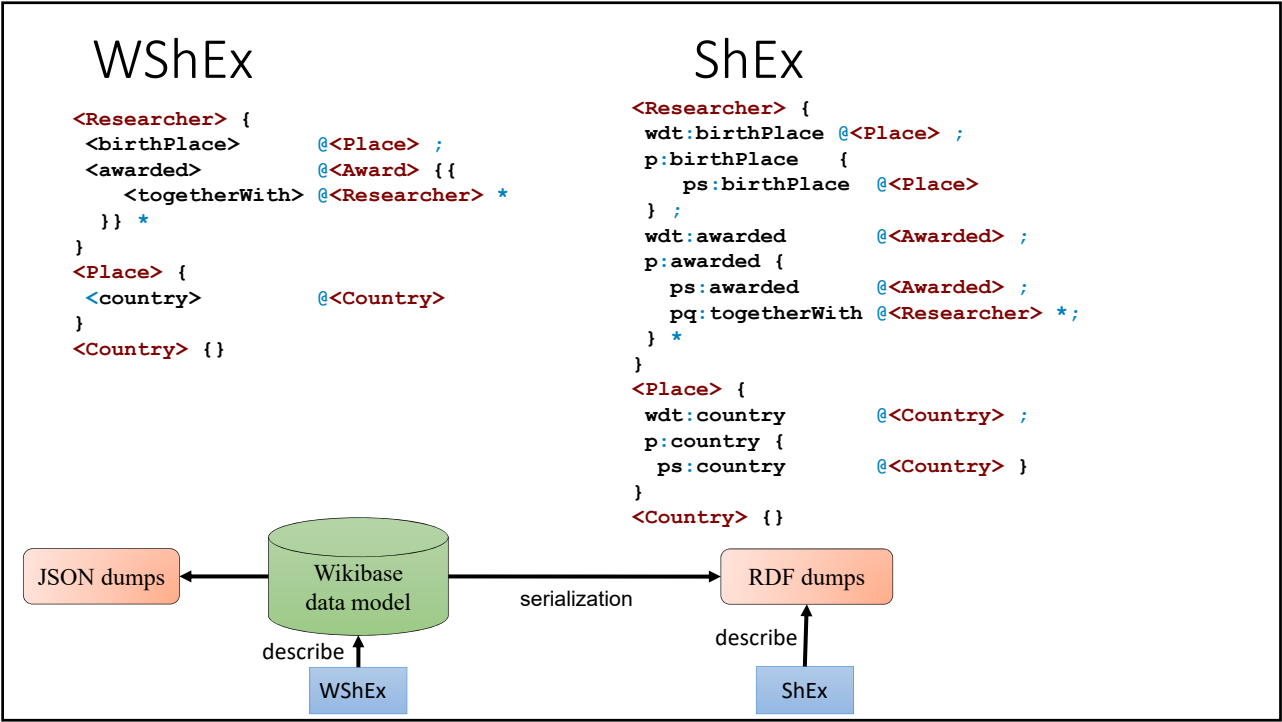


\mathcal{L}	$= \{ \text{Person, Place, Country, Organization, Date, Award} \}$
$\delta(\text{Person})$	$= \{ \neg \overrightarrow{\text{birthDate}} @Date; \neg \overrightarrow{\text{birthPlace}} @Place; \neg \overrightarrow{\text{employer}} @Organization [start : @Date, end : @Date]^*; \neg \overrightarrow{\text{awarded}} @Award [pointTime : @Date, togetherWith : @Person]^* \}$
$\delta(\text{Place})$	$= \{ \neg \overrightarrow{\text{country}} @Country \}$
$\delta(\text{Country})$	$= \{ \}$
$\delta(\text{Award})$	$= \{ \neg \overrightarrow{\text{country}} @Country \}$
$\delta(\text{Organization})$	$= \{ \}$
$\delta(\text{Date})$	$= \{ \in xsd : date \}$

What's the role of WShEx

- WShEx schemas describe Wikibase data model
 - Closer to JSON dumps
- ShEx (entity schemas) describe RDF serialization of Wikibase data model



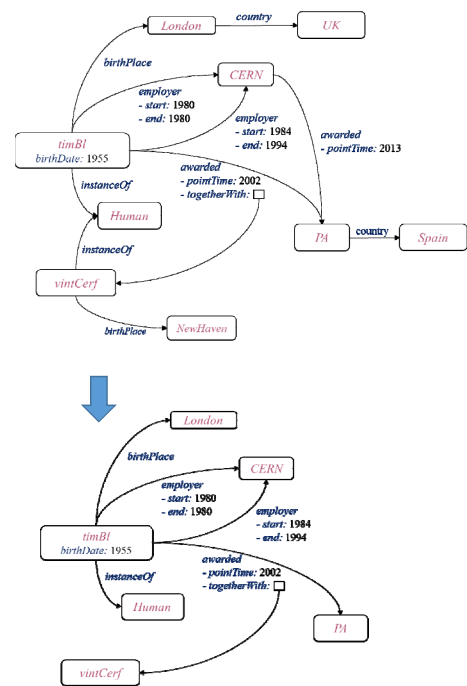


Entity-matching

Directly matches an entity like an ítem, property, etc.

Example: $Q_s = \{timBl\}$

- $Q' = \{timBl, CERN, vintCerf, PA\}$
- $\mathcal{P}' = \{birthDate, birthPlace, employer, awarded, start, end, togetherWith\}$
- $\mathcal{D}' = \{1984, 1994, 1980, 1955\}$
- $S' = \{ (timBl, birthDate, 1955, \{\}), (timBl, birthPlace, London, \{\}), (timBl, employer, CERN, \{start : 1980, end : 1980\}), (timBl, employer, CERN, \{start : 1984, end : 1994\}), (timBl, awarded, PA, \{togetherWith : vintCerf\}), (vintCerf, awarded, PA, \{togetherWith : timBl\}) \}$



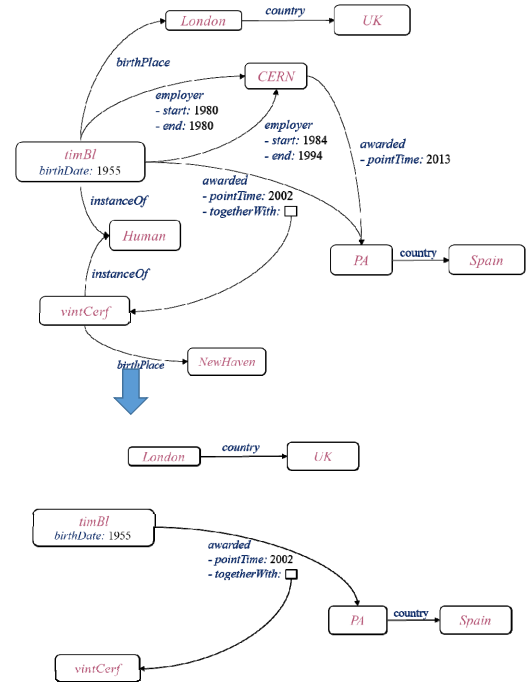
Simple matching

Defines a matching language:

$m ::=$	subject(e)	Subject $e \in \mathcal{E}$
	property(p)	Property $p \in \mathcal{P}$
	value(v)	Value $v \in \mathcal{V}$
	qualifier(p, v)	Qualifier with property $p \in \mathcal{P}$ and value $v \in \mathcal{V}$
	qualifiedProp(p)	Qualifier with property $p \in \mathcal{P}$
	qualifiedValue(v)	Qualifier with value $v \in \mathcal{V}$

Example: $M_s = \{property(country), qualifiedProp(togetherWith)\}$

- $Q' = \{PA, Spain, London, UK, timBl, vintCerf\}$
- $\mathcal{P}' = \{country, awarded, togetherWith\}$
- $\mathcal{D}' = \{\}$
- $S' = \{ (timBl, awarded, PA, \{togetherWith : vintCerf\}), (PA, country, Spain, \{\}), (London, country, UK, \{\}) \}$

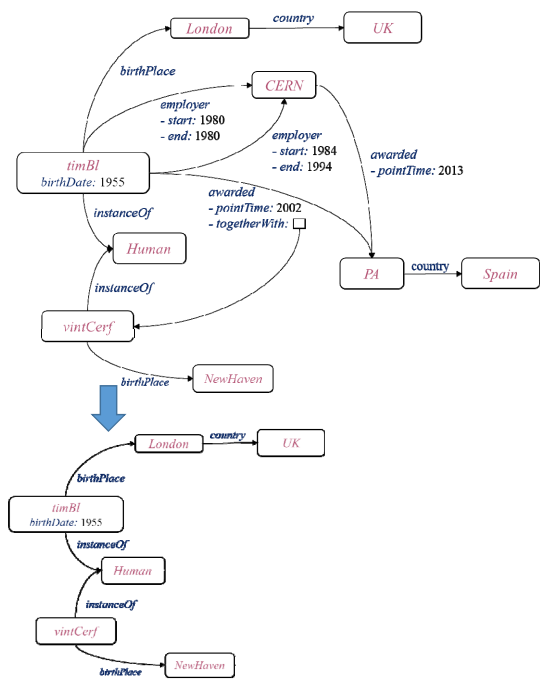


ShEx-based matching

Uses ShEx definitions to match entities
Replaces shape references by *true*
Doesn't require graph traversal

```

L = { Researcher, Place, Country, Date, Human }
δ(Researcher) = {
  - instanceOf → @Human;
  - birthDate → @Date?;
  - birthPlace → @Place
}
δ(Place) = {
  - country → @Country
}
δ(Date) = {
  - ∈ xsd:date
}
δ(Human) = {
  - ∈ {Human}
}
S = {
  (timBl, instanceOf, Human, {}),
  (timBl, birthDate, 1955, {}),
  (timBl, birthPlace, London, {}),
  (London, country, UK, {}),
  (vintCerf, instanceOf, Human, {}),
  (vintCerf, birthPlace, newHaven, {})
}
```



ShEx-based matching

Can be implemented replacing shape references by *true*

```

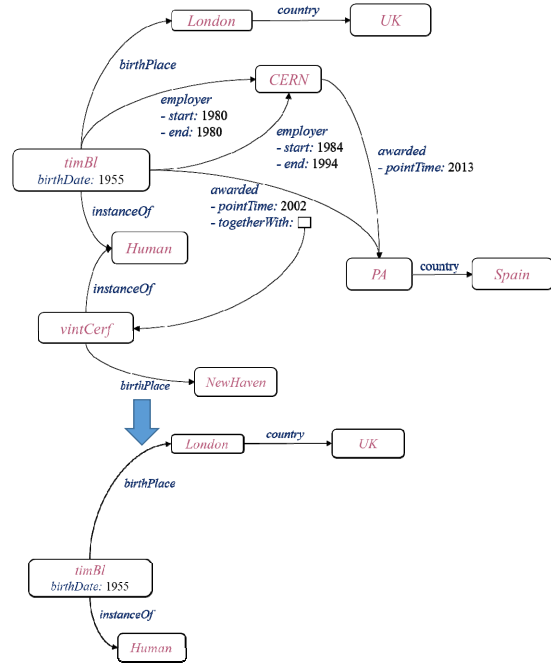
δ(Researcher) = {
  - instanceOf → @Human;
  - birthDate → @Date?;
  - birthPlace → @Place
}
δ(Place) = {
  - country → @Country
}
δ(Date) = {
  - ∈ xsd:date
}
δ(Human) = {
  - ∈ {Human}
}
→
δ(Researcher) = {
  - instanceOf → true;
  - birthDate → true?;
  - birthPlace → true
}
δ(Place) = {
  - country → true
}
```

ShEx-based matching has been implemented in WDSub
It allows sequential processing of the dump
It can process the latest Wikidata dump in 5h 15'

ShEx+slurp

Slurp = Shex option that collects nodes/triples while validating

Example:

$$\begin{aligned} \mathcal{L} &= \{ \text{Researcher, Place, Country, Date, Human} \} \\ \delta(\text{Researcher}) &= \{ \begin{array}{l} \text{instanceOf} \rightarrow @Human; \\ \text{birthDate} \rightarrow @Date; \\ \text{birthPlace} \rightarrow @Place \end{array} \} \\ \delta(\text{Place}) &= \{ \begin{array}{l} \text{country} \rightarrow @Country \end{array} \} \\ \delta(\text{Country}) &= \{ \} \\ \delta(\text{Date}) &= \in xsd : date \\ \delta(\text{Human}) &= \in \{ Human \} \end{aligned}$$


ShEx+Slurp

Formal definition generalizes ShEx semantics

New conformance relation:

$$\begin{aligned} \text{Cond} & \frac{\text{cond}(n) = \text{true}}{\mathcal{G}, n, \tau \models \text{cond} \rightsquigarrow \langle \{n\}, \{ \} \rangle} \quad \text{AND} \frac{\mathcal{G}, n, \tau \models se_1 \rightsquigarrow \mathcal{G}_1 \quad \mathcal{G}, n, \tau \models se_2 \rightsquigarrow \mathcal{G}_2}{\mathcal{G}, n, \tau \models se_1 \text{ AND } se_2 \rightsquigarrow \mathcal{G}_1 \cup \mathcal{G}_2} \\ \text{ClosedShape} & \frac{\text{neighs}(n, \mathcal{G}) = ts \quad \mathcal{G}, ts, \tau \models s' \rightsquigarrow \mathcal{G}'}{\mathcal{G}, n, \tau \vdash \text{CLOSED } s' \rightsquigarrow \mathcal{G}'} \\ \text{OpenShape} & \frac{ts = \{ \langle x, p, y \rangle \mid \langle x, p, y \rangle \in \text{neighs}(n, \mathcal{G}) \mid p \in \text{preds}(tc) \} \quad \mathcal{G}, ts, \tau \models s' \rightsquigarrow \mathcal{G}'}{\mathcal{G}, n, \tau \models s' \rightsquigarrow \mathcal{G}'} \\ \text{EachOf} & \frac{(ts_1, ts_2) \in \text{part}(ts) \quad \mathcal{G}, ts_1, \tau \models te_1 \rightsquigarrow \mathcal{G}_1 \quad \mathcal{G}, ts_2, \tau \models te_2 \rightsquigarrow \mathcal{G}_2}{\mathcal{G}, ts, \tau \models te_1; te_2 \rightsquigarrow \mathcal{G}_1 \cup \mathcal{G}_2} \\ \text{OneOf}_1 & \frac{\mathcal{G}, ts, \tau \models te_1 \rightsquigarrow \mathcal{G}_1}{\mathcal{G}, ts, \tau \models te_1 \mid te_2 \rightsquigarrow \mathcal{G}_1} \quad \text{OneOf}_2 \frac{\mathcal{G}, ts, \tau \models te_2 \rightsquigarrow \mathcal{G}_2}{\mathcal{G}, ts, \tau \models te_1 \mid te_2 \rightsquigarrow \mathcal{G}_2} \\ \text{Star}_1 & \frac{}{\mathcal{G}, \emptyset, \tau \models te^* \rightsquigarrow \emptyset} \\ \text{Star}_2 & \frac{(ts_1, ts_2) \in \text{part}(ts) \quad \mathcal{G}, ts_1, \tau \models te \rightsquigarrow \mathcal{G}_1 \quad \mathcal{G}, ts_2, \tau \models te \rightsquigarrow \mathcal{G}_2}{\mathcal{G}, ts, \tau \models te^* \rightsquigarrow \mathcal{G}_1 \cup \mathcal{G}_2} \\ \text{TripleConstraint} & \frac{ts = \{ \langle x, p, y, s \rangle \} \quad \mathcal{G}, y, \tau \models @l \rightsquigarrow \langle \mathcal{V}, \mathcal{E} \rangle \quad \mathcal{G}, s, \tau \models qs \rightsquigarrow (qs', \mathcal{G}_{qs})}{\mathcal{G}, ts, \tau \models \text{ } \xrightarrow{p} @l qs \rightsquigarrow \langle \mathcal{V} \cup \{x\} \cup \{y\}, \mathcal{E} \cup \langle x, p, y, qs' \rangle \rangle \cup \mathcal{G}_{qs}} \\ \text{OpenQs} & \frac{s' = \{ \langle p, v \rangle \in s \mid p \in \text{preds}(ps) \} \quad \mathcal{G}, s', \tau \models ps \rightsquigarrow (qs, \mathcal{G}')}{\mathcal{G}, s, \tau \vdash [ps] \rightsquigarrow (qs, \mathcal{G}')} \\ \text{CloseQs} & \frac{\mathcal{G}, s, \tau \vdash ps \rightsquigarrow (qs, \mathcal{G}')}{\mathcal{G}, s, \tau \vdash [ps] \rightsquigarrow (qs, \mathcal{G}')} \\ \text{EachOfQs} & \frac{\mathcal{G}, s, \tau \vdash ps_1 \rightsquigarrow (qs_1, \mathcal{G}_1) \quad \mathcal{G}, s, \tau \vdash ps_2 \rightsquigarrow (qs_2, \mathcal{G}_2)}{\mathcal{G}, s, \tau \vdash ps_1, ps_2 \rightsquigarrow (qs_1 \cup_2, \mathcal{G}_1 \cup \mathcal{G}_2)} \\ \text{OneOfQs}_1 & \frac{\mathcal{G}, s, \tau \vdash ps_1 \rightsquigarrow (qs_1, \mathcal{G}_1)}{\mathcal{G}, s, \tau \vdash ps_1 \mid ps_2 \rightsquigarrow (qs_1, \mathcal{G}_1)} \quad \text{OneOfQs}_2 \frac{\mathcal{G}, s, \tau \vdash ps_2 \rightsquigarrow (qs_2, \mathcal{G}_2)}{\mathcal{G}, s, \tau \vdash ps_1 \mid ps_2 \rightsquigarrow (qs_2, \mathcal{G}_2)} \\ \text{StarQs}_1 & \frac{}{\mathcal{G}, \emptyset, \tau \vdash ps^* \rightsquigarrow \{ \}, \emptyset} \\ \text{StarQs}_2 & \frac{(s_1, s_2) \in \text{part}(s) \quad \mathcal{G}, s_1, \tau \vdash ps \rightsquigarrow (qs_1, \mathcal{G}_1) \quad \mathcal{G}, s_2, \tau \vdash ps \rightsquigarrow qs_2, \mathcal{G}_2}{\mathcal{G}, s, \tau \vdash ps^* \rightsquigarrow (qs_1 \cup qs_2, \mathcal{G}_1 \cup \mathcal{G}_2)} \\ \text{EmptyQs} & \frac{}{\mathcal{G}, \emptyset, \tau \vdash \epsilon \rightsquigarrow \{ \}, \emptyset} \quad \text{PropertyQs} \frac{s = \{ \langle p, v \rangle \} \quad \mathcal{G}, v, \tau \models @l \rightsquigarrow \mathcal{G}'}{\mathcal{G}, s, \tau \vdash p : @l \rightsquigarrow \{ \langle p : v \rangle, \mathcal{G}' \}} \end{aligned}$$

ShEx+Slurp

Implemented by shex.js and pyshex

Only for ShEx, not for PShEx or WShEx

It does graph traversal while validating

Difficult to use for processing Wikidata dumps

Subsets include only valid data according to the ShEx

Problems:

Difficult to run behind SPARQL endpoint

Naïve implementation generates too many requests

Optimizations to improve number of SPARQL queries can generate timeouts

ShEx+Pregel

Large scale validation

Based on Pregel algorithm

Developed by Google 2010

Based on Bulk Synchronous Parallel model

Supersteps = units of parallel computation

Communication through messages

Implemented using Apache Spark GraphX library

Think like a vertex

Vertex-centric approach

Each vertex has an id and a state

All vertexes receive an initial message

While messages received do

Run `vProg` to update vertex state according to messages received

Run `sendMsg` for each triplet to generate messages

Merge messages by Vertex

GraphX

Graph API from Apache Spark

RDD (Resilient distributed datasets)

MapReduce in memory

$Graph[\mathcal{V}, \mathcal{E}]$ = graph with vertices \mathcal{V} and edges \mathcal{E}

- Vertices are represented as $RDD[(Id, \mathcal{V})]$ where $Id = Long$
- Edges are represented as $RDD[(Id, Id, \mathcal{E})]$
- *triplets* view represents edges as collections $RDD[(\mathcal{V}, \mathcal{E}, \mathcal{V})]$.

Several built-in operators:

- `mapVertices(g: Graph[\mathcal{V}, \mathcal{E}], f: (Id, \mathcal{V}) \rightarrow \mathcal{V}): Graph[\mathcal{V}, \mathcal{E}]`
- `mapReduceTriples(g: Graph[\mathcal{V}, \mathcal{E}], m: ($\mathcal{V}, \mathcal{E}, \mathcal{V}$) \rightarrow [(Id, \mathcal{M})], r: (\mathcal{M}, \mathcal{M}) \rightarrow \mathcal{M}): RDD[(Id, \mathcal{M})]`
- `joinVertices(g: Graph[\mathcal{V}, \mathcal{E}], msgs: RDD[(Id, \mathcal{M})], f: ($Id, \mathcal{V}, \mathcal{M}$) \rightarrow \mathcal{V}): Graph[\mathcal{V}, \mathcal{E}]`
- `pregel` (see next slide)
- ...

GraphX Pregel pseudo-code

Algorithm 1: Pregel algorithm pseudocode as implemented in GraphX

Input parameters:

g : Graph[\mathcal{V}, \mathcal{E}]
 initialMsg : \mathcal{M}
 vProg : $(\text{Id}, \mathcal{V}, \mathcal{M}) \rightarrow \mathcal{V}$
 sendMsg : Triplet $\rightarrow [(\text{Id}, \mathcal{M})]$
 mergeMsg : $(\mathcal{M}, \mathcal{M}) \rightarrow \mathcal{M}$

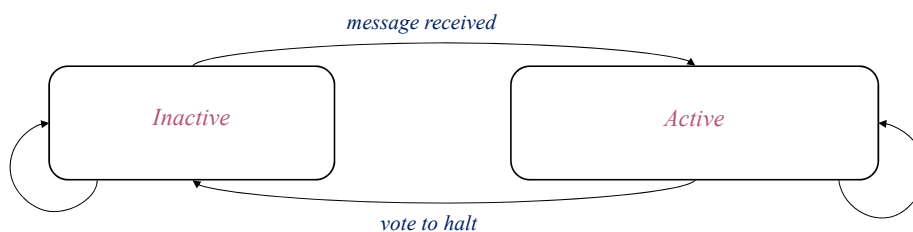
Output: g : Graph[\mathcal{V}, \mathcal{E}]

```

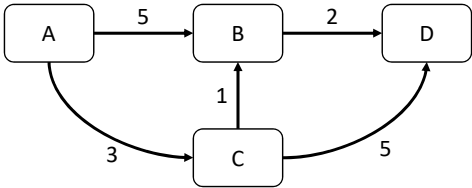
1  $g = \text{mapVertices}(g, \lambda(\text{id}, v) \rightarrow \text{vProg}(\text{id}, v, \text{initialMsg}))$ 
2  $\text{msgs} = \text{mapReduceTriples}(g, \text{sendMsg}, \text{mergeMsg})$ 
3 while  $\text{size}(\text{msgs}) > 0$  do
4    $g = \text{joinVertices}(g, \text{msgs}, \text{vProg})$ 
5    $\text{msgs} = \text{mapReduceTriples}(g, \text{sendMsg}, \text{mergeMsg})$ 
6 return  $g$ 

```

Pregel's vertex state diagram



Shortest path to a node



```
val initialGraph = graph.mapVertices(
  (id, _) => if (id == source) 0 else ∞
)

initialMsg = ∞

def vProg(id: VertexId, dist: Int, msg: Int) =
  min(dist, msg)

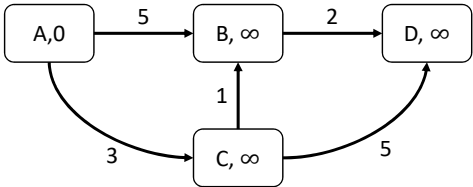
def sendMsg(t: Triplet) = {
  if (t.srcAttr + t.attr < t.dstAttr) {
    (t.dstId, t.srcAttr + t.attr)
  }
  else { }
}

def mergeMsg(a: Int, b: Int) = min(a, b)

initialGraph.pregel(initialMsg)
                (vProg, sendMsg, mergeMsg)
```

Shortest path to a node

initialGraph



```
val initialGraph = graph.mapVertices(
  (id, _) => if (id == source) 0 else ∞
)

initialMsg = ∞

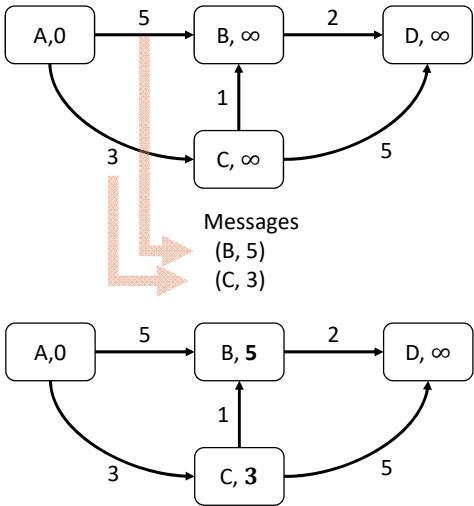
def vProg(id: VertexId, dist: Int, msg: Int) =
  min(dist, msg)

def sendMsg(t: Triplet) = {
  if (t.srcAttr + t.attr < t.dstAttr) {
    (t.dstId, t.srcAttr + t.attr)
  }
  else { }
}

def mergeMsg(a: Int, b: Int) = min(a, b)

initialGraph.pregel(initialMsg)
                (vProg, sendMsg, mergeMsg)
```


Shortest path to a node



```
val initialGraph = graph.mapVertices(
  (id, _) => if (id == source) 0 else ∞
)

initialMsg = ∞

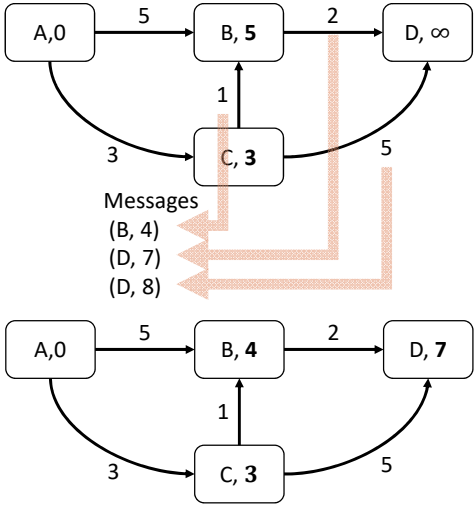
def vProg(id: VertexId, dist: Int, msg: Int) =
  min(dist, msg)

def sendMsg(t: Triplet) = {
  if (t.srcAttr + t.attr < t.dstAttr) {
    (t.dstId, t.srcAttr + t.attr)
  }
  else { }
}

def mergeMsg(a: Int, b: Int) = min(a, b)

initialGraph.pregel(initialMsg)
                (vProg, sendMsg, mergeMsg)
```

Shortest path to a node



```
val initialGraph = graph.mapVertices(
  (id, _) => if (id == source) 0 else ∞
)

initialMsg = ∞

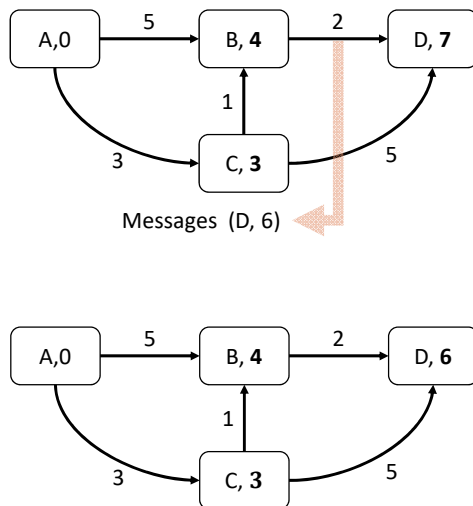
def vProg(id: VertexId, dist: Int, msg: Int) =
  min(dist, msg)

def sendMsg(t: Triplet) = {
  if (t.srcAttr + t.attr < t.dstAttr) {
    (t.dstId, t.srcAttr + t.attr)
  }
  else { }
}

def mergeMsg(a: Int, b: Int) = min(a, b)

initialGraph.pregel(initialMsg)
                (vProg, sendMsg, mergeMsg)
```

Shortest path to a node



```
val initialGraph = graph.mapVertices(
  (id, _) => if (id == source) 0 else ∞
)

initialMsg = ∞

def vProg(id: VertexId, dist: Int, msg: Int) =
  min(dist, msg)

def sendMsg(t: Triplet) = {
  if (t.srcAttr + t.attr < t.dstAttr) {
    (t.dstId, t.srcAttr + t.attr)
  }
  else { }
}

def mergeMsg(a: Int, b: Int) = min(a, b)

initialGraph.pregel(initialMsg)
  (vProg, sendMsg, mergeMsg)
```

Idea: Validate graphs using Pregel?

Associate each node with a status map

Status map contains information about node conformance to shapes

Messages =

- Requests to validate
- Request to wait for neighbours
- Information about validated/failed neighbours

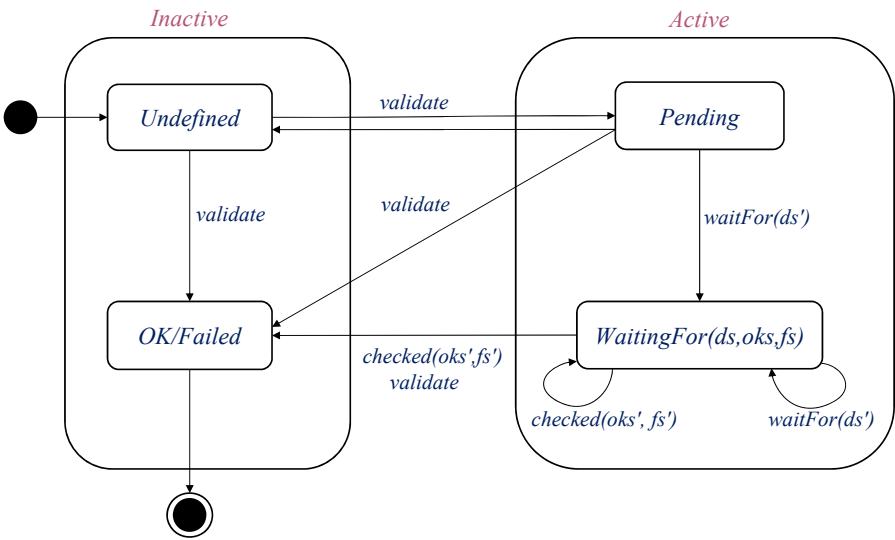
Initial message = request to validate start shape

In each iteration we collect information from neighbours conformance

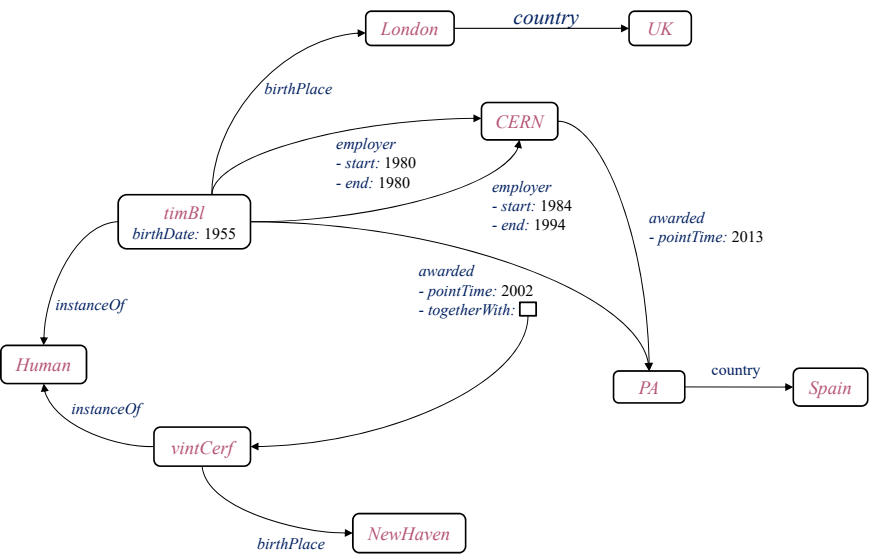
We provide 3 parameters

- checkLocal: tries to check conformance of a node locally or return pending neighbours
- checkNeighs: checks neighbours with regular expression defined by a shape
- tripleConstraints: return the triple constraints associated with a shape label

State diagram



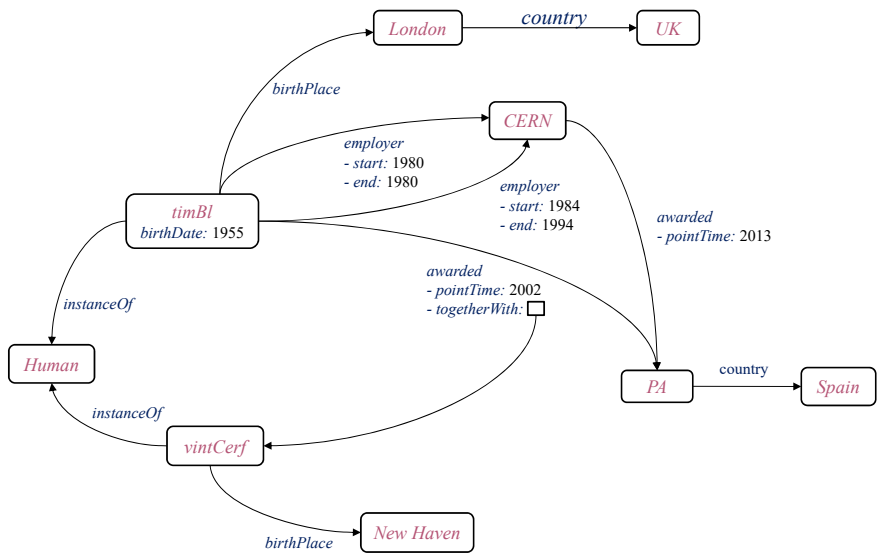
Trace of PSchema



WShEx
Schema

```
start = <Researcher> {  
  instanceOf @<Human> ;  
  birthDate   Date? ;  
  birthPlace  @<Place> ;  
}  
<Human> [ Human ]  
<Place> {  
  country @<Country>  
}  
<Country> { }
```

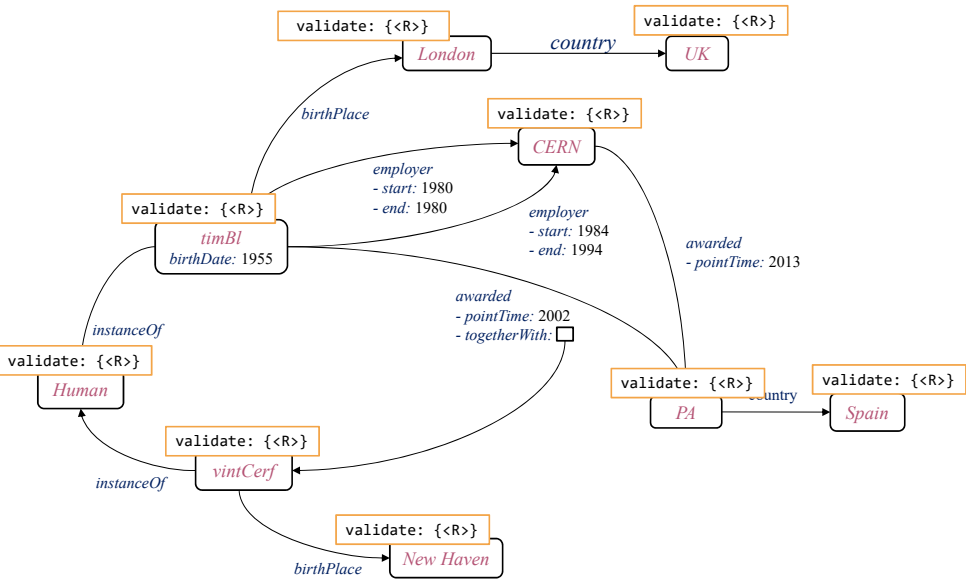
Trace of PSchema



WSHEx Schema

```
start = <R> {
  instanceOf @<H> ;
  birthDate Date? ;
  birthPlace @<P> ;
}
<H> [ Human ]
<P> {
  country @<C>
}
<C> { }
```

Superstep 0. All nodes asked to validate start = <R>



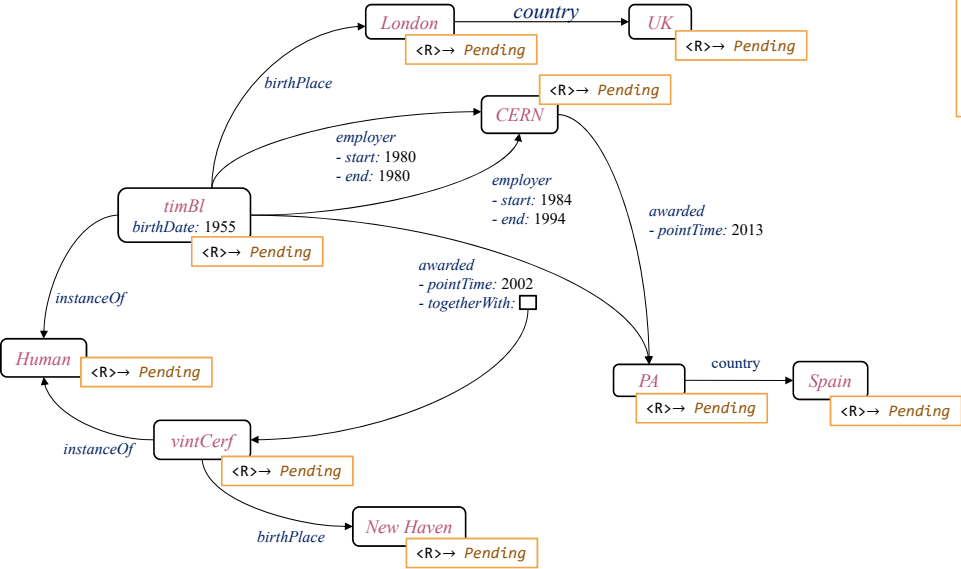
Schema

```
start = <R> {
  instanceOf @<H> ;
  birthDate Date? ;
  birthPlace @<P> ;
}
<H> [ Human ]
<P> {
  country @<C>
}
<C> { }
```

Messages at superstep 0

```
(timBl, validate: <R>),
(London, validate: <R>),
(UK, validate: <R>),
(CERN, validate: <R>),
(PA, validate: <R>),
(Spain, validate: <R>),
(Human, validate: <R>),
(vintCerf, validate: <R>),
(newHaven, validate: <R>)
```

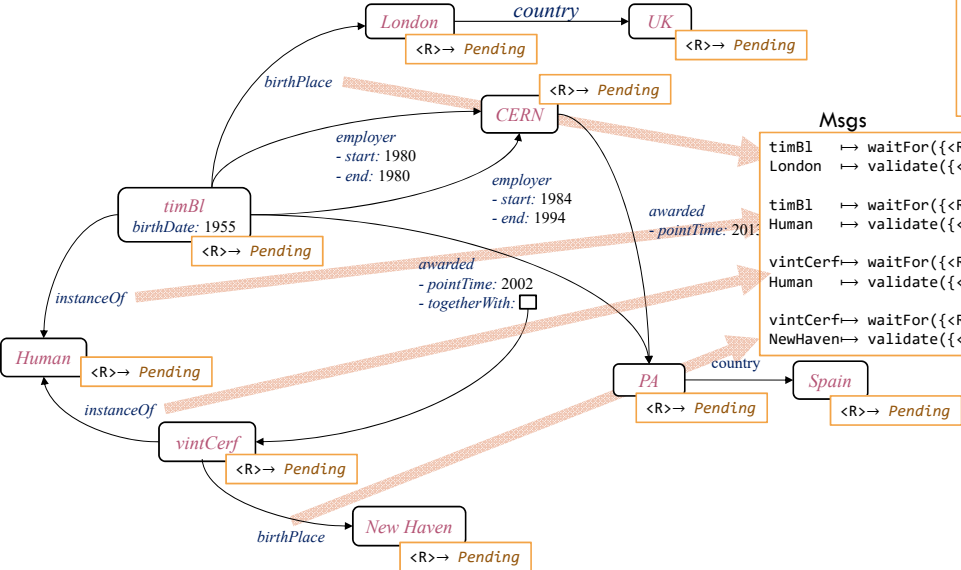
Superstep 0. After vprog



Schema

```
start = <R> {
  instanceOf @<H> ;
  birthDate Date? ;
  birthPlace @<P> ;
}
<H> [ Human ]
<P> {
  country @<C>
}
<C> { }
```

Superstep 1. Collecting messages



Schema

```
start = <R> {
  instanceOf @<H> ;
  birthDate Date? ;
  birthPlace @<P> ;
}
<H> [ Human ]
<P> {
  country @<C>
}
<C> { }
```

Msgs

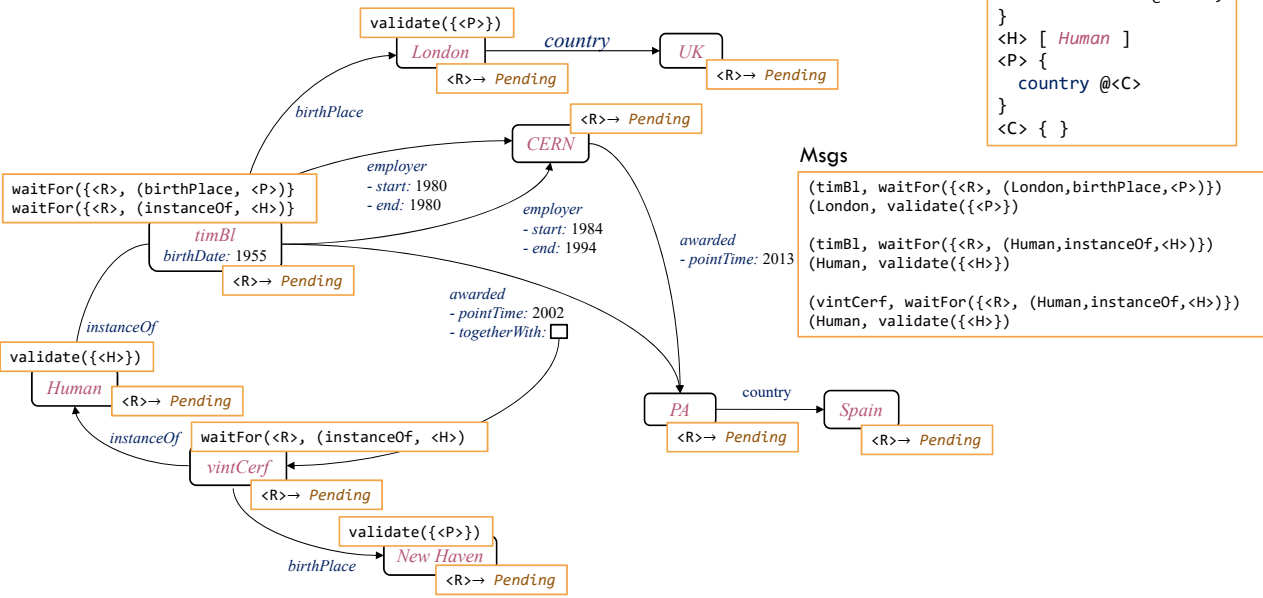
```
timBl  => waitFor({<R>, (London, birthPlace, <P>)})
London => validate({<P>})

timBl  => waitFor({<R>, (Human, instanceOf, <H>)})
Human  => validate({<H>})

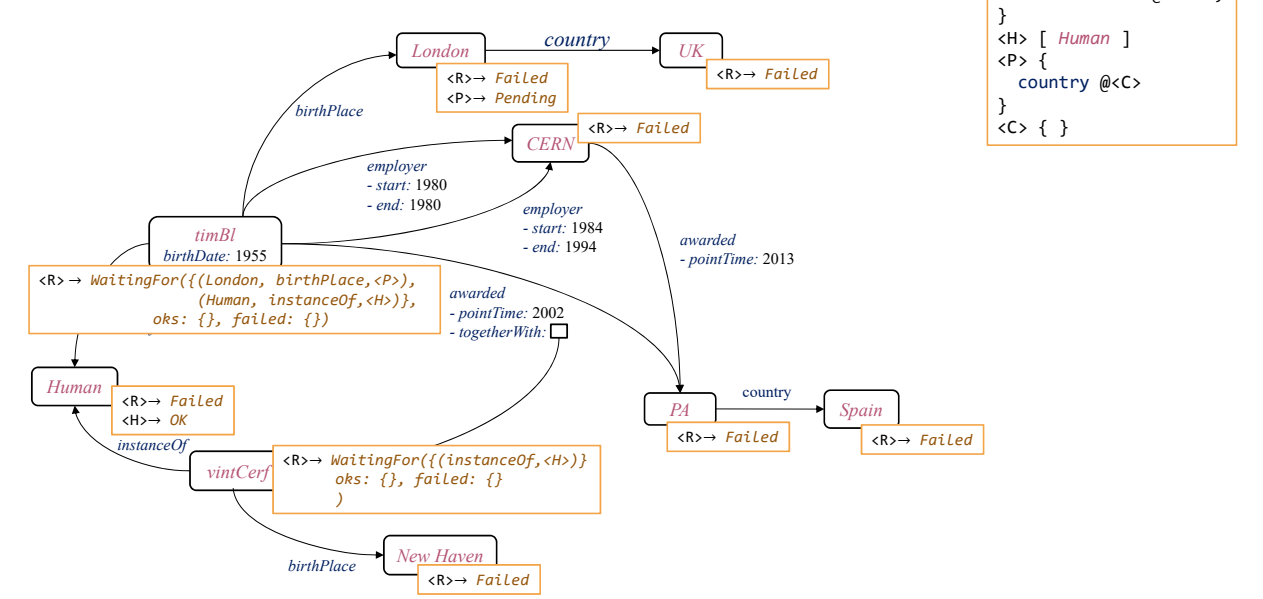
vintCerf => waitFor({<R>, (Human, instanceOf, <H>)})
Human  => validate({<H>})

vintCerf => waitFor({<R>, (NewHaven, instanceOf, <H>)})
NewHaven => validate({<P>})
```

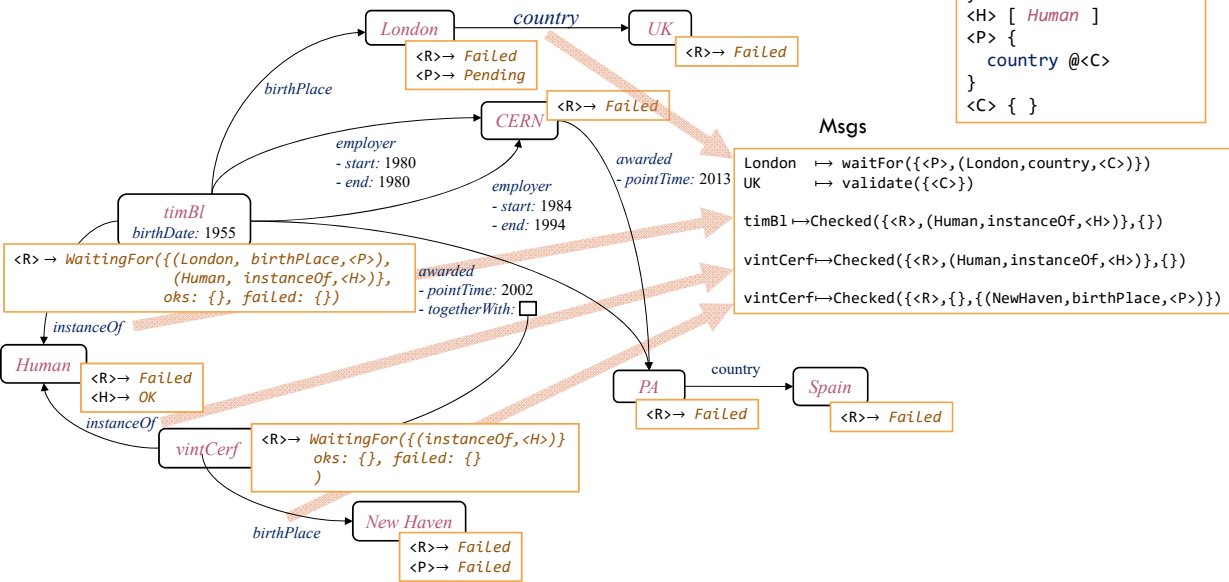
Superstep 1. Collecting messages



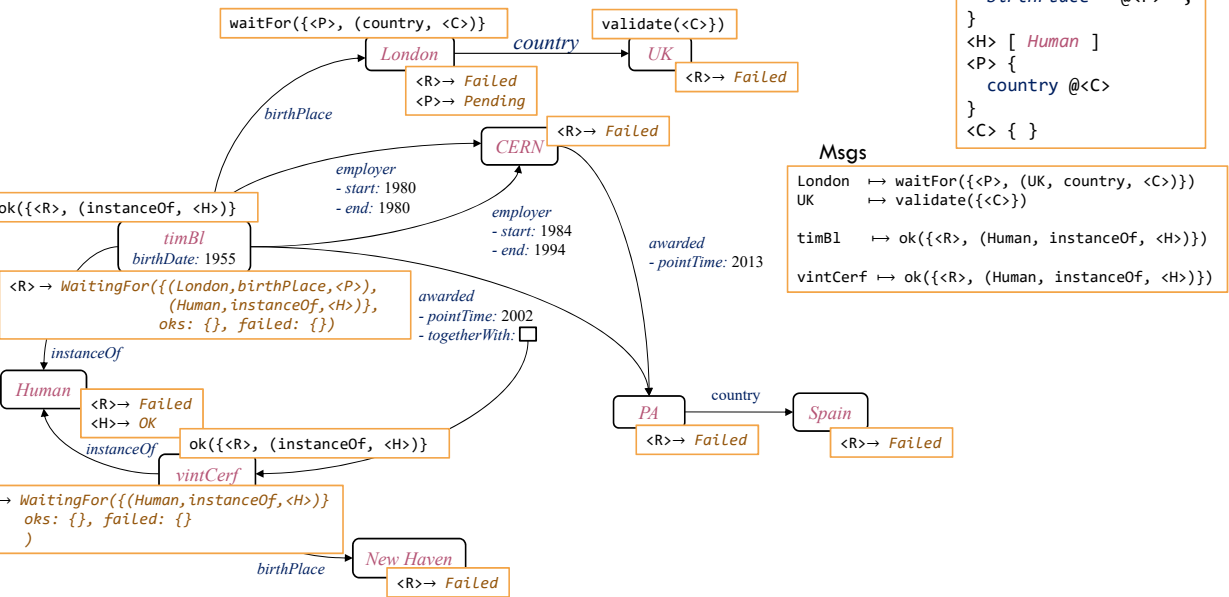
Superstep 1. After vprog.



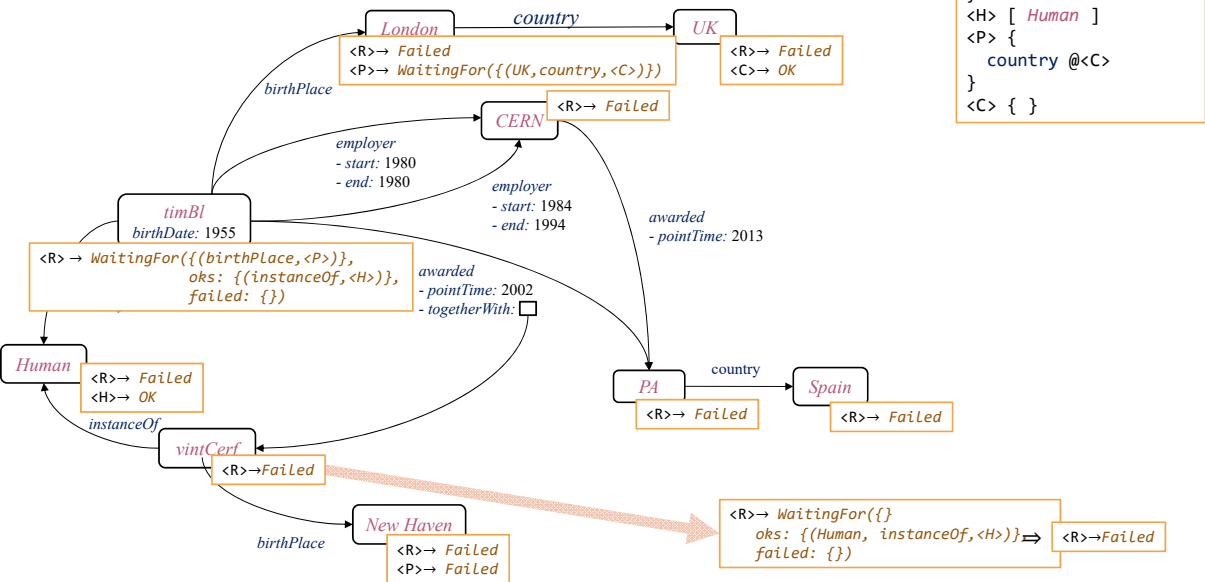
Superstep 2. Collecting new messages.



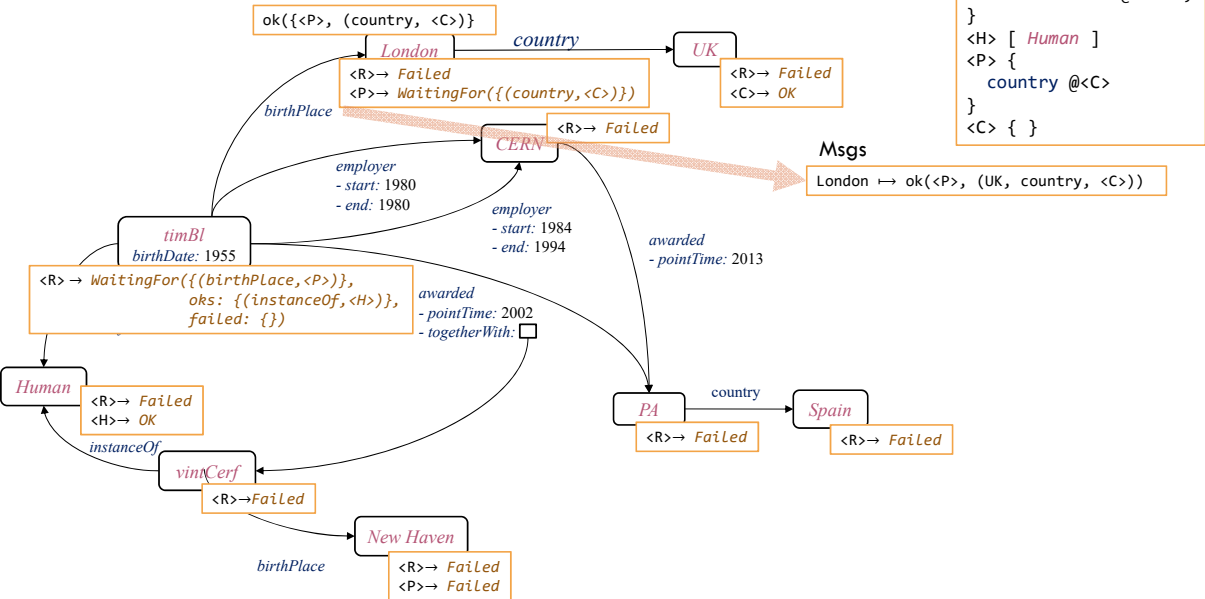
Superstep 2. After vprog



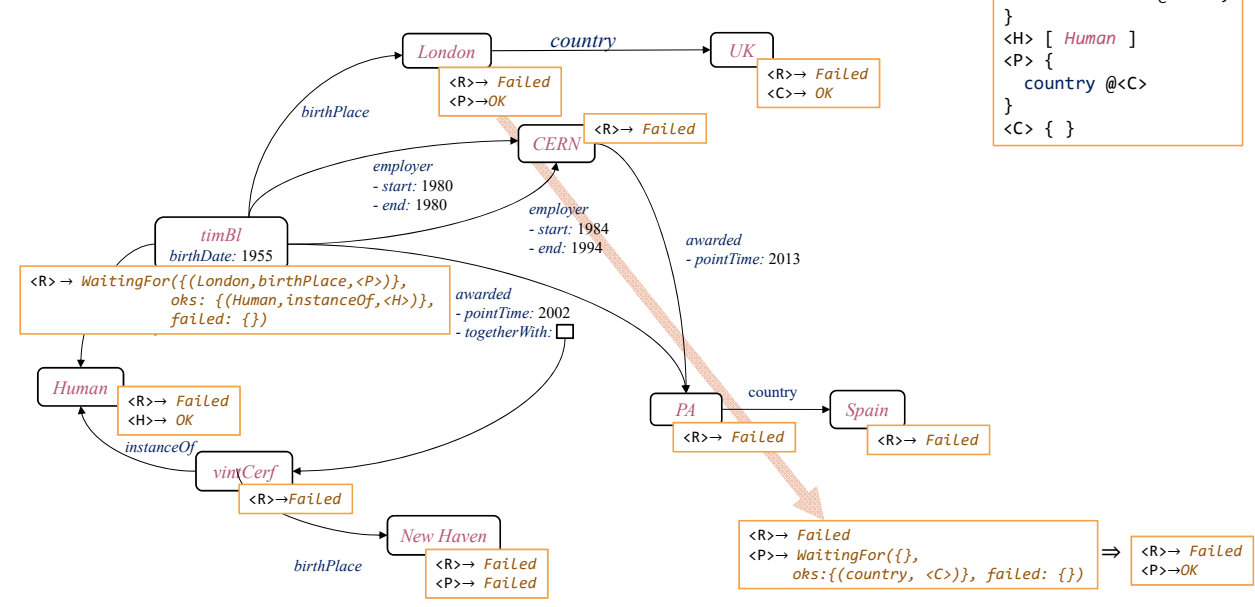
Superstep 2. After vprog



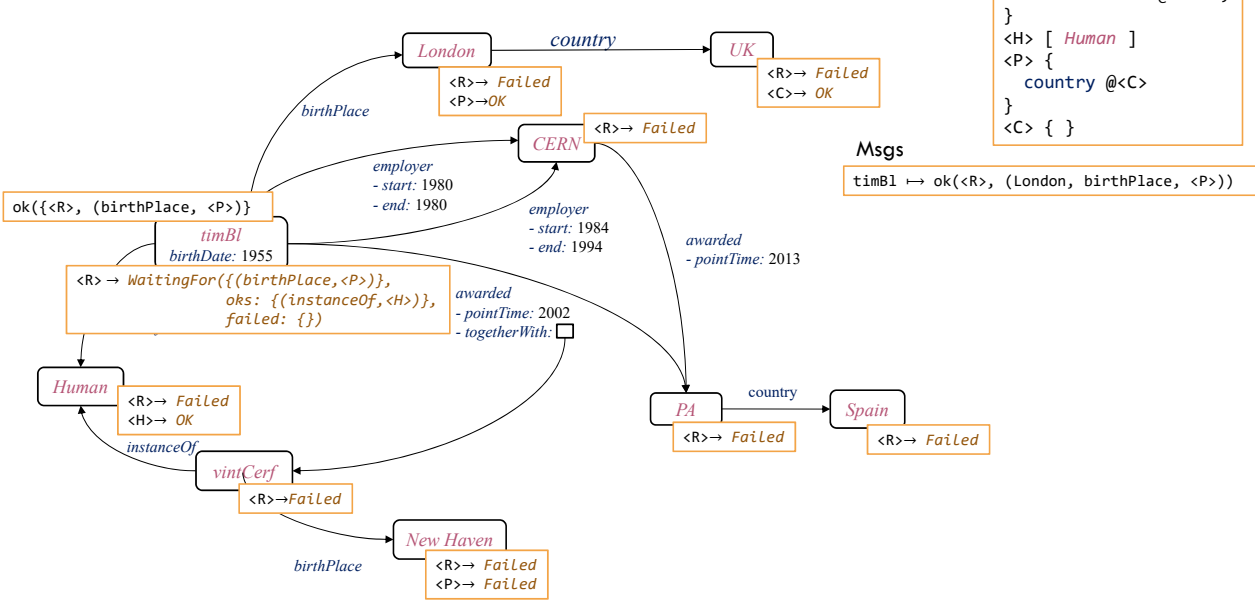
Superstep 3. Collecting messages



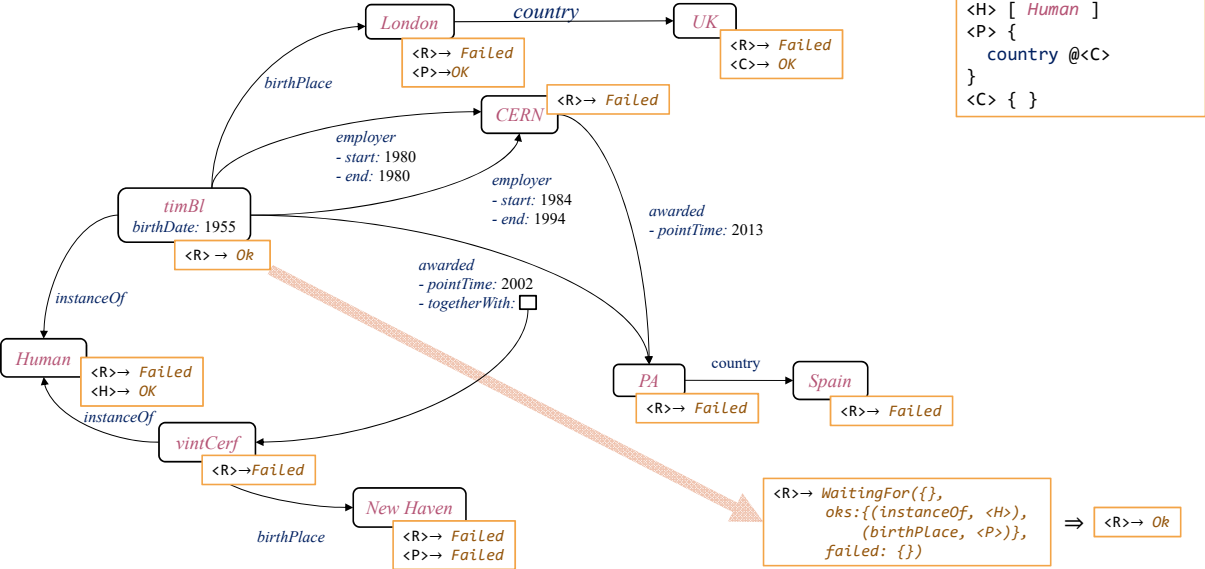
Superstep 3. After vprog



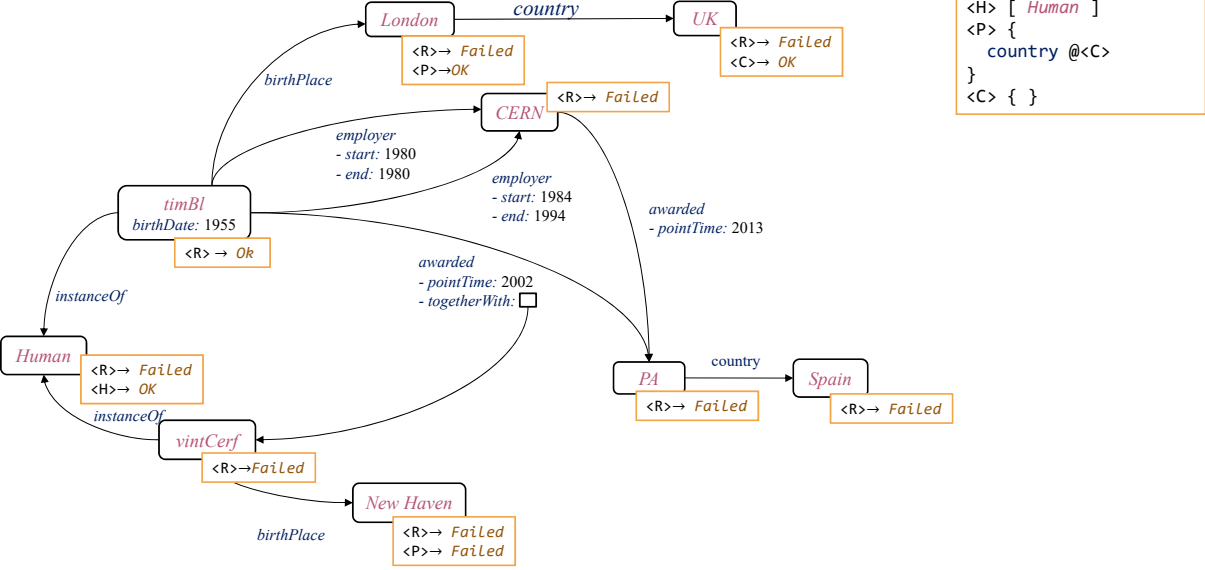
Superstep 4. Collecting messages



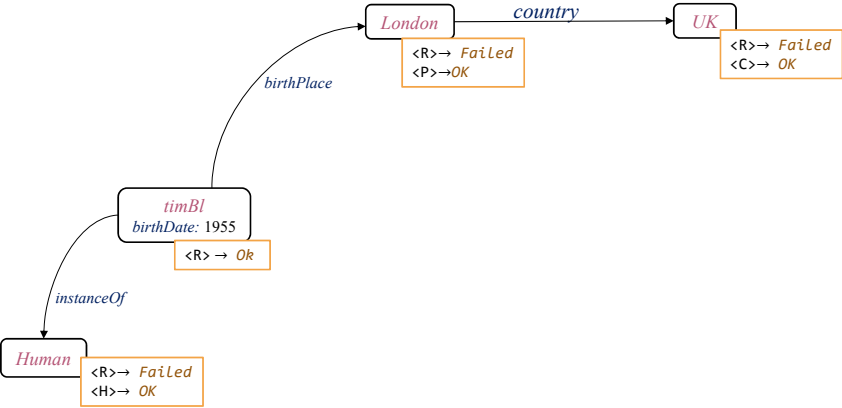
Superstep 4. After vprog



End state: All nodes inactive, no more messages



Wikibase Subgraph: entities with some OK shape



Schema

```
start = <R> {
  instanceOf @<H> ;
  birthDate Date? ;
  birthPlace @<P> ;
}
<H> [ Human ]
<P> {
  country @<C>
}
<C> { }
```

Status and Messages

<i>Status</i>	::=	<i>Undefined</i>	Default status
		<i>Ok</i>	Node conforms
		<i>Failed</i>	Node doesn't conform
		<i>Pending</i>	Requested to conform
		<i>WaitingFor</i> (<i>ds</i> , <i>oks</i> , <i>fs</i>)	Waiting for some neighbours <i>ds</i> = list dependants neighbours <i>oks</i> = list of conformant neighbours <i>fs</i> = list of non conformant neighbours where <i>ds</i> , <i>oks</i> , <i>failed</i> ∈ $\mathcal{V} \times \mathcal{P} \times \mathcal{L}$
<i>Msg</i>	::=	<i>Validate</i>	Request to validate
		<i>Checked</i> (<i>oks</i> , <i>fs</i>)	Some neighbours have been checked <i>oks</i> = neighbours that have been checked as conformant <i>fs</i> = neighbours that have been checked as non-conformant where <i>oks</i> , <i>fs</i> ∈ $\mathcal{V} \times \mathcal{P} \times \mathcal{L}$
		<i>WaitFor</i> (<i>ds</i>)	Request to wait for some neighbours where <i>ds</i> ∈ $\mathcal{V} \times \mathcal{P} \times \mathcal{L}$

Adapting Pregel to validate graphs using ShEx

Algorithm 1: Pregel-based ShEx validation pseudocode

Input parameters:

```

g: Graph[V, E]
initialLabel: L
checkLocal: (L, V) → Ok | Failed | Pending(Set[L])
checkNeighs: (L, Bag[(E, L)], Set[(E, L)]) → Ok | Failed
tripleConstraints: L → Set[(E, L)]

```

Output: g: Graph[V, L → Status, E]

```
gs = mapVertices(g, λ(id, v) → (id, (v, λv → Undefined)))
```

```
gs = pregel(Validate, gs, vProg, sendMsg, mergeMsg)
```

```
gs = mapVertices(gs, checkUnsolved)
```

```
return gs
```

```
def checkUnsolved(v, m) = (v, m') where
```

```
m'(l) =
```

```

{ checkNeighs(l, ∅, ∅)           if m(l) = Pending
{ checkNeighs(l, oks, fs ∪ ds)   if m(l) = WaitingFor(ds, oks, fs)
{ m(l)                           otherwise

```

Parameters

InitialLabel, start label

checkLocal, attempts to check if a node conforms locally. Possible results:

Ok

Failed

Pending(ls)

checkNeighs, attempts to check the neighbourhood of a node. Result:

Ok

Failed

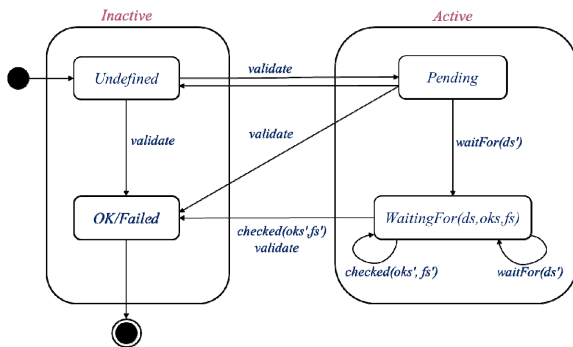
tripleConstraints: set of constraints associated with a label

vProg definition

$vProg(id, (n, m), msg) = (n, m')$ where

$m'(l) = m(l)$

except for the cases indicated by the rules:



$$\frac{(n, m), l \rightsquigarrow \text{Validate} \quad m(l) = s \in \{\text{Undefined}, \text{Pending}\} \quad \text{checkLocal}(l, n) = r \in \{\text{Ok}, \text{Failed}\}}{m'(l) = r}$$

$$\frac{(n, m), l \rightsquigarrow \text{Validate} \quad m(l) = r \in \{\text{Undefined}, \text{Pending}\} \quad \text{checkLocal}(l, n) = \text{Pending}(ls)}{m'(l) = \text{Undefined} \quad m'(l') = \text{Pending} \forall l' \in ls}$$

$$\frac{(n, m), l \rightsquigarrow \text{Validate} \quad m(l) = r \in \{\text{Ok}, \text{Failed}\}}{m'(l) = r}$$

$$\frac{(n, m), l \rightsquigarrow \text{Validate} \quad m(l) = \text{WaitingFor}(ds, oks, fs)}{m'(l) = \text{Ok}}$$

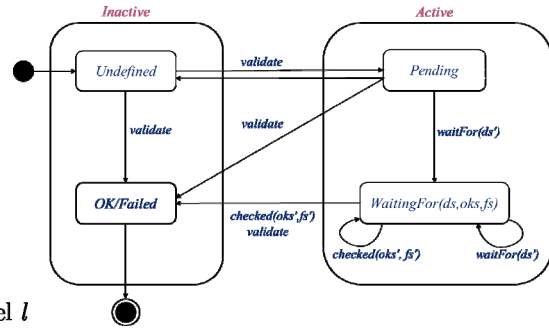
$$\frac{(n, m), l \rightsquigarrow \text{Checked}(oks, fs) \quad m(l) = \text{WaitingFor}(ds, oks', fs') \quad ds \setminus (oks \cup fs) \neq \emptyset}{m'(l) = \text{WaitingFor}(ds, oks \cup oks', fs \cup fs')}$$

$$\frac{(n, m), l \rightsquigarrow \text{Checked}(oks, fs) \quad m(l) = \text{WaitingFor}(ds, oks', fs') \quad ds \setminus (oks \cup fs) = \emptyset}{m'(l) = \text{checkNeighs}(l, oks \cup oks', fs \cup fs')}$$

sendMsg

$\langle (s, m_s), p, (o, m_o) \rangle =$
 triplet view with
 - subject (s, m_s)
 - predicate p
 - object (o, m_o)

$(x, m_x), l \rightsquigarrow Msg =$
 message Msg sent to node x with status map m_x for label l



$$\frac{\langle (s, m_s), p, (o, m_o) \rangle \in \mathcal{G} \quad m_s(l) = Pending \quad tcs(l, \mathcal{S}) = \perp \xrightarrow{p} @l'}{(s, m_s), l \rightsquigarrow WaitFor((o, p, l')) \quad (o, m_o), l \rightsquigarrow Validate}$$

$$\frac{\langle (s, m_s), p, (o, m_o) \rangle \in \mathcal{G} \quad m_s(l) = WaitFor(ds, oks, fs) \quad (o, p, l') \in ds \quad m_o(l') = Ok}{(s, m_s), l \rightsquigarrow Checked((o, p, l'), \emptyset)}$$

$$\frac{\langle (s, m_s), p, (o, m_o) \rangle \in \mathcal{G} \quad m_s(l) = WaitFor(ds, oks, fs) \quad (o, p, l') \in ds \quad m_o(l') = Failed}{(s, m_s), l \rightsquigarrow Checked(\emptyset, (o, p, l'))}$$

mergeMsg

$$mergeMsg((n, m), l \rightsquigarrow msg_1, (n, m), l \rightsquigarrow msg_2) = (n, m), l \rightsquigarrow msg_1 \oplus msg_2$$

$$\begin{aligned} Validate \oplus y &= y \\ Validate \oplus Checked(oks, fs) &= Checked(oks, fs) \\ Validate \oplus WaitFor(ds) &= WaitFor(ds) \\ Checked(oks, fs) \oplus Validate &= Checked(oks, fs) \\ Checked(oks, fs) \oplus Checked(oks', fs') &= Checked(oks \cup oks', fs \cup fs') \\ Checked(oks, fs) \oplus WaitFor(ds) &= Checked(oks \cup ds, fs \cup fs) \\ WaitFor(ds) \oplus Validate &= WaitFor(ds) \\ WaitFor(ds) \oplus Checked(oks, fs) &= Checked(oks \cup ds, fs) \\ WaitFor(ds) \oplus WaitFor(ds') &= WaitFor(ds \cup ds') \end{aligned}$$

checkLocal $\text{checkLocal}: (\mathcal{L}, \mathcal{V}) \rightarrow \text{Ok} \mid \text{Failed} \mid \text{Pending}(\text{Set}[\mathcal{L}])$

Checks if it is possible to validate a node locally

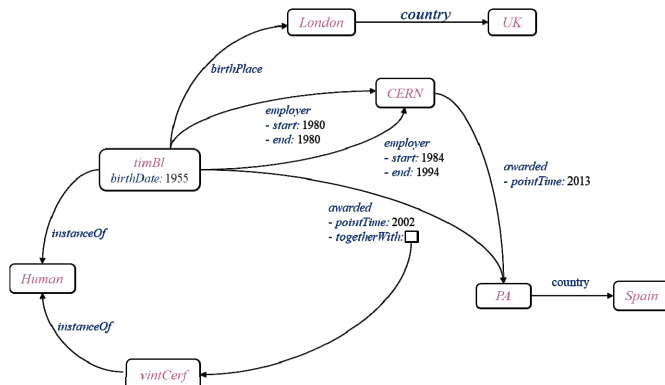
Returns

Ok: Node conforms

Failed: Node doesn't conform

Pending(ls) conformance depends on neighbours

```
checkLocal(<Human>, Human) = Ok
checkLocal(<Researcher>, Human) = Failed
checkLocal(<Researcher>, timBL) = Pending { <Human>, <Place> }
```



```
start = <Researcher>
<Researcher> {
  instanceof @<Human> ;
  birthDate Date? ;
  birthPlace @<Place> ;
}
<Human> [ Human ]
<Place> {
  country @<Country>
}
<Country> { }
```

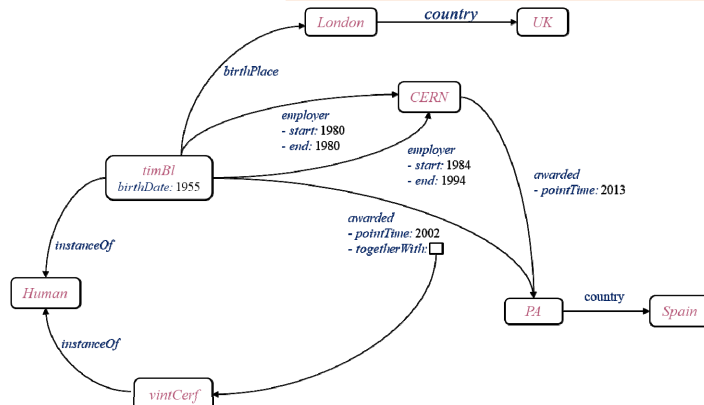
checkNeighs $\text{checkNeighs}: (\mathcal{L}, \text{Bag}[(\mathcal{E}, \mathcal{L})], \text{Set}[(\mathcal{E}, \mathcal{L})]) \rightarrow \text{Ok} \mid \text{Failed}$

Checks if the bag of neighbours of a node conform to the regular bag expression defined by a shape

Returns: ok (conforms), failed (doesn't)

```
checkNeighs(<Researcher>, {(instanceOf, <Human>), (birthPlace, <London>)}, {}) = Ok
checkNeighs(<Researcher>, {(instanceOf, <Human>) }, {}) = Failed
```

```
rbe(<Researcher>) = (instanceOf, <Human>);
(birthPlace, <Place>)
```



```
start = <Researcher>
<Researcher> {
  instanceof @<Human> ;
  birthDate Date? ;
  birthPlace @<Place> ;
}
<Human> [ Human ]
<Place> {
  country @<Country>
}
<Country> { }
```

tripleConstraints

$\text{tripleConstraints}: \mathcal{L} \rightarrow \text{Set}[(\mathcal{E}, \mathcal{L})]$

Returns the triple constraints of a shape label

```
tripleConstraints(<Researcher>) = {(instanceOf, <Human>), (birthPlace, <Place>)}
tripleConstraints(<Human>) = {}
tripleConstraints(<Place>) = {(country, <Country>)}
tripleConstraints(<Country>) = {}
```

```
start = <Researcher>
<Researcher> {
  instanceOf @<Human> ;
  birthDate Date? ;
  birthPlace @<Place> ;
}
<Human> [ Human ]
<Place> {
  country @<Country>
}
<Country> { }
```

Preliminary results

Implemented in SparkWDSUB using Apache Spark GraphX

No optimizations applied

Latest Wikidata dumps on AWS can be processed on AWS

512 cores, 3.904 Gb RAM, 121.600 Gb disk

For 2014 Wikidata dump (31.3 GB uncompressed) in 3 minutes

For 2021 Wikidata dump (1.256,55 Gb uncompressed) it took 36 minutes

Only for simple schemas yet

Conclusions

- Formal definition of Knowledge graphs
 - RDF graphs, property graphs, wikibase graphs
- ShEx extension for property graphs/wikibase graphs
- Formal definition of wikibase graphs subsets
- Scalable approach to validate big knowledge graphs
 - Inspired by Pregel algorithm (Spark GraphX)

Repaso de contribuciones

	Grafos RDF	Property graphs	Grafos Wikibase
Definición formal	Realizado previamente	Realizado previamente	Presente trabajo
Descripción grafos conocimiento	ShEx	PShEx	WShEx
	Realizado anteriormente	Presente trabajo	Presente trabajo
Entity-generated (definición)	-	-	Presente trabajo
Entity-generated (implementación)	-	-	WDumper
Simple matching (definición)	-	-	Presente trabajo
Simple matching (implementación)	-	-	Wdumper
ShEx-based matching (definición)	-	-	Presente trabajo
ShEx-based matching (implementación)	-	-	WDSUB
ShEx+Slurp (definición)	-	-	Presente trabajo
ShEx+Slurp (implementación)	ShEx.js, pyShEx	-	-
ShEx+Pregel (definición)	-	-	Presente trabajo
ShEx+Pregel (implementación)	-	-	SparkWDSUB

Fin presentación prueba 2

Jose Emilio Labra Gayo



Universidad de Oviedo