The work presented in this talk is based on discussions with the following colleagues (in alphabetical order): Alejandro Flores, Bryan Thompson, Grant Weddell, Juan Sequeda, Kavitha Srinivas, Maria-Esther Vidal, Mike Personick, Orri Erling, Peter Boncz, Tamer Özsu, and Yrjänä Rankka.
Outline

1. The Data Models
2. Property Graphs to RDF
3. RDF to Property Graphs
A Property Graph

Properties

Key      Value

name = "Stanley Kubrick" birthyear = 1928

certainty = 0.8

influencedBy

Kubrick

Welles

mentioned

Node

Label

Edge

name = "Orson Welles"
An RDF “Graph”

RDF Triple

( http://example.name/alice , http://xmlns.com/foaf/0.1/name , "Alice" )
( http://example.name/bob , http://xmlns.com/foaf/0.1/name , "Bob" )
( http://example.name/bob , http://xmlns.com/foaf/0.1/age , 23 )

Subject  
Predicate  
Object
An RDF “Graph”

prefix foaf: http://xmlns.com/foaf/0.1/
prefix ex: http://example.name/

( ex:alice, foaf:knows, ex:bob )
( ex:alice, foaf:name, "Alice" )
( ex:bob, foaf:name, "Bob" )
( ex:bob, foaf:age, 23 )
An RDF “Graph”

prefix foaf: http://xmlns.com/foaf/0.1/
prefix ex: http://example.name/

( ex:alice , foaf:knows , ex:bob )
( ex:alice , foaf:name , "Alice" )
( ex:bob , foaf:name , "Bob" )
( ex:bob , foaf:age , 23 )
Statement-level Metadata?

prefix foaf: http://xmlns.com/foaf/0.1/
prefix ex: http://example.name/

( ex:alice , foaf:knows , ex:bob )
( ex:alice , foaf:name , "Alice" )
( ex:bob , foaf:name , "Bob" )
( ex:bob , foaf:age , 23 )

certainty = 0.8
RDF Reification

prefix foaf: http://xmlns.com/foaf/0.1/
prefix ex: http://example.name/

(ex:alice, foaf:knows, ex:bob)
(ex:alice, foaf:name, "Alice")
(ex:bob, foaf:name, "Bob")
(ex:bob, foaf:age, 23)
(_,b1, rdf:type, rdf:Statement)
(_,b1, rdf:subject, ex:alice)
(_,b1, rdf:predicate, foaf:knows)
(_,b1, rdf:object, ex:bob)
(_,b1, ex:certainty, 0.8)
Querying with SPARQL

SELECT ?c
WHERE {
  ?s rdf:type rdf:Statement .
  ?s rdf:subject ex:alice .
  ?s rdf:predicate foaf:knows .
  ?s rdf:object ex:bob .
  ?s ex:certainty ?c
}
SELECT ?c
WHERE {
  ?s   rdf:type   rdf:Statement .
  ?s   rdf:subject   ex:alice .
  ?s   rdf:predicate   foaf:knows .
  ?s   rdf:object   ex:bob .
  ?s   ex:certainty   ?c .
  ex:alice   foaf:known   ex:bob
}
RDF*

prefix foaf: http://xmlns.com/foaf/0.1/
prefix ex: http://example.name/

(( ex:alice , foaf:knows , ex:bob ) , ex:certainty , 0.8 )
( ex:alice , foaf:name , "Alice" )
( ex:bob , foaf:name , "Bob" )
( ex:bob , foaf:age , 23 )
RDF*

prefix foaf: http://xmlns.com/foaf/0.1/
prefix ex: http://example.name/

( ( ex:alice , foaf:knows , ex:bob ) , ex:certainty , 0.8 )
( ex:alice , foaf:name , "Alice" )
( ex:bob , foaf:name , "Bob" )
( ex:bob , foaf:age , 23 )

Metadata triple
SPARQL* 

SELECT ?c 
WHERE { 
  <<ex:alice foaf:knows ex:bob>> ex:certainty ?c 
}
SPARQL*

SELECT ?c
WHERE {
  <<ex:alice foaf:knows ex:bob>> ex:certainty ?c
}

SELECT ?c
WHERE {
  ?s rdf:type rdf:Statement .
  ?s rdf:subject ex:alice .
  ?s rdf:predicate foaf:knows .
  ?s rdf:object ex:bob .
  ?s ex:certainty ?c .
  ex:alice foaf:known ex:bob
}
Outline

1. The Data Models ✓
2. Property Graphs to RDF
3. RDF to Property Graphs
Outline

1. The Data Models ✓
2. Property Graphs to RDF*
3. RDF* to Property Graphs
PGs to RDF* Preliminaries

- Assume a **vertex identity mapping**, e.g., \texttt{Kubrick} → \_:b1
- Assume an **edge label mapping**, e.g., mentioned → http://example.org/relationship/mentioned
- Assume a **property key mapping**, e.g., name → http://example.org/property/name
- Assume a **value to literal mapping**

- certainty = 0.8
- name = "Stanley Kubrick"
- birthyear = 1928
- influencedBy
- name = "Orson Welles"
PGs to RDF* Idea

- Transform each edge (+ its label) to an ordinary triple
- Transform each node property into an ordinary triple
- Transform each edge property to a metadata triple
PGs to RDF* Example

name = "Stanley Kubrick"
birthday = 1928

influencedBy

name = "Orson Welles"

prefix p: http://example.org/property/
prefix r: http://example.org/relationship/

( _:b1 , p:name , "Stanley Kubrick")
( _:b1 , p:birthday , 1928 )
( _:b2 , p:name , "Orson Welles" )
( _:b2 , r:mentioned , _:b1 )
( ( _:b1 , r:influencedBy, _:b2 ) , p:certainty , 0.8 )
Querying the Resulting RDF*

```
SELECT ?n WHERE {
  ?w p:name "Orson Welles" .
}
ORDER BY ?c
```

( _:b1 , p:name , "Stanley Kubrick" )
( _:b1 , p:birthyear , 1928 )
( _:b2 , p:name , "Orson Welles" )
( _:b2 , r:mentioned , _:b1 )
( ( _:b1 , r:influencedBy , _:b2 ) , p:certainty , 0.8 )
“Equivalent” Cypher Query

```
SELECT ?n WHERE {
  ?w p:name "Orson Welles" .
}
ORDER BY ?c
```

```
START p=node(*)
MATCH (p)-[x:influencedBy]->( w { name="Orson Welles" } )
RETURN p.name
ORDER BY x.certainty
```
Limitation

- Distinct edges with the same source node, the same target node, and the same label would be mapped to a single RDF triple
  - RDF graphs / RDF* graphs are sets

\[ \text{certainty} = 0.8 \]
\[ \text{influencedBy} \]

\[ \text{certainty} = 0.5 \]
Limitation

- Distinct edges with the same source node, the same target node, and the same label would be mapped to a single RDF triple
  - RDF graphs / RDF* graphs are sets

```
Kubrick
  influencedBy
    certainty = 0.8
  
Welles
  influencedBy
    certainty = 0.5
```
Outline

1. The Data Models ✓
2. Property Graphs to RDF* ✓
3. RDF* to Property Graphs
RDF* to PGs Idea

- Transform each ordinary triple to an edge
- Transform each metadata triple to an edge property
RDF* to PGs Example

( ( ex:alice , foaf:knows , ex:bob ) , ex:certainty , 0.8 )
( ex:alice , foaf:name , "Alice" )
( ex:bob , foaf:name , "Bob" )
( ( ex:bob , foaf:age , 23 ) , ex:certainty , 0.5 )
RDF* to PGs Example

((ex:alice, foaf:knows, ex:bob), ex:certainty, 0.8)
((ex:alice, foaf:name, "Alice")
((ex:bob, foaf:name, "Bob")
(((ex:bob, foaf:age, 23), ex:certainty, 0.5))
RDF* to PGs Example

( ( ex:alice , foaf:knows , ex:bob ) , ex:certainty , 0.8 )
( ex:alice , foaf:name , "Alice" )
( ex:bob , foaf:name , "Bob" )
( ( ex:bob , foaf:age , 23 ) , ex:certainty , 0.5 )
RDF* to PGs Example

((ex:alice, foaf:knows, ex:bob), ex:certainty, 0.8)
(ex:alice, foaf:name, "Alice")
(ex:bob, foaf:name, "Bob")
(((ex:bob, foaf:age, 23), ex:certainty, 0.5)
PG-Convertible RDF* Graphs

Condition 1: Metadata triples are not nested

\[
\text{( (ex:alice,foaf:knows,ex:bob) , ex:certainty, 0.8 )}
\]
PG-Convertible RDF* Graphs

Condition 1: Metadata triples are not nested

( (ex:alice, foaf:knows, ex:bob) , ex:certainty, 0.8 ) ,

ex:date, 2014-11-14 )

\[ v_1 \xrightarrow{\text{foaf:knows}} v_2 \]

kind = "IRI"
IRI = "ex:alice"

kind = "IRI"
IRI = "ex:bob"

ex:certainty = 0.8

ex:date = 2014-11-14
PG-Convertible RDF* Graphs

Condition 1: Metadata triples are not nested

\[
( ( \text{ex:alice}, \text{foaf:knows}, \text{ex:bob} ), \text{ex:certainty}, 0.8 ), \\
\text{ex:date}, 2014-11-14 )
\]
PG-Convertible RDF* Graphs

Condition 1: Metadata triples are not nested
Condition 2: Triples embedded as subject only

( ex:olaf , ex:believes , (ex:alice,foaf:knows,ex:bob) )
PG-Convertible RDF* Graphs

Condition 1: Metadata triples are not nested
Condition 2: Triples embedded as subject only

( ex:olaf , ex:believes , (ex:alice,foaf:knows,ex:bob) )
PG-Convertible RDF* Graphs

Condition 1: Metadata triples are not nested
Condition 2: Triples embedded as subject only
Condition 3: Object of any metadata triple must be a literal

( (ex:alice, foaf:knows, ex:bob) , ex:source , ex:olaf )
PG-Convertible RDF* Graphs

Condition 1: Metadata triples are not nested

Condition 2: Triples embedded as subject only

Condition 3: Object of any metadata triple must be a literal

\[(\text{ex:alice, foaf:knows, ex:bob}, \text{ex:source, ex:olaf})\]
PG-Convertible RDF* Graphs

Condition 1: Metadata triples are not nested
Condition 2: Triples embedded as subject only
Condition 3: Object of any metadata triple must be a literal
Condition 4: Any literal must be convertible to a value of some (programming language specific) data type
Properties of the Mapping

- **Lossless**: any Property Graph produced by the mapping contains all information present in the original RDF* graph
  - Resulting Property Graphs are “RDF-like”
- For PG users, such Property Graphs may be “unnatural” and too complex
  - Nodes for literals
  - Queries become quite verbose
Example Queries

SELECT ?pn WHERE {
   ?a foaf:name "Alice" .
   ?p foaf:name ?pn  }

START a=node(*)
MATCH (a)-[:foaf:name]->( bn { literal="Alice" } ) ,
   (a)-[:foaf:knows]->(p)-[:foaf:name]->(pn)
RETURN pn.literal
Example Queries

SELECT ?pn WHERE {
    ?a foaf:name "Alice" .
    ?p foaf:name ?pn
}

START a=node(*)
MATCH  (a)[:-][foaf:name]->( bn { literal="Alice" } ) ,
       (a)[:-][foaf:knows]->(p)[:-][foaf:name]->(pn)
RETURN pn.literal

START a=node(*)
MATCH  (a {foaf:name="Alice"})[:-][foaf:knows]->(p)
RETURN p.foaf:name
Outline

1. The Data Models ✓
2. Property Graphs to RDF* ✓
3. RDF* to
   … “RDF like” Property Graphs ✓
   … “Simple” Property Graphs
RDF* to “Simple” PGs

( ( ex:alice , foaf:knows , ex:bob ) , ex:certainty , 0.8 )
( ex:alice , foaf:name , "Alice" )
( ex:bob , foaf:name , "Bob" )
( ex:bob , foaf:age , 23 )

1. Transform each relationship triple to an edge
   – i.e., ordinary triples with an IRI or bnode object
RDF* to “Simple” PGs

((ex:alice, foaf:knows, ex:bob), ex:certainty, 0.8)
((ex:alice, foaf:name, "Alice")
((ex:bob, foaf:name, "Bob")
((ex:bob, foaf:age, 23))

1. Transform each *relationship triple* to an edge
   - i.e., ordinary triples with an IRI or bnode object

\[
\begin{align*}
V_1 \quad \text{IRI} = \text{"ex:alice"} \\
\text{foaf:knows} \quad \rightarrow \\
V_2 \quad \text{IRI} = \text{"ex:bob"}
\end{align*}
\]
RDF* to “Simple” PGs

((ex:alice, foaf:knows, ex:bob), ex:certainty, 0.8)
((ex:alice, foaf:name, "Alice")
((ex:bob, foaf:name, "Bob")
((ex:bob, foaf:age, 23)

2. Transform *attribute triples* to node properties
   - i.e., ordinary triples with a literal as object
RDF* to “Simple” PGs

((ex:alice, foaf:knows, ex:bob), ex:certainty, 0.8)
(ex:alice, foaf:name, "Alice")
(ex:bob, foaf:name, "Bob")
(ex:bob, foaf:age, 23)

3. Transform each metadata triple about a relationship triple to an edge property
Strong PG-Convertibility

Condition 1: PG-convertibility

Condition 2: No metadata triple about an attribute triple

\[
( (\text{ex:}bob, \text{foaf:}age, 23), \text{ex:}certainty, 0.5 )
\]
Strong PG-Convertibility

Condition 1: PG-convertibility
Condition 2: No metadata triple about an attribute triple

\[(\text{ex:bob, foaf:age, 23}, \text{ex:certainty, 0.5})\]
Summary

- **RDF* – an compact form of RDF reification**
  - Turtle*, SPARQL*

- **Mappings:**
  - PG-convertible RDF* to “RDF-like” PGs (lossless)
  - strongly PG-convertible RDF* to “simple” PGs
  - trivial (any RDF graph is an RDF* graph) by “unfolding” metadata triples

- **Documents:** arxiv/1409.3288 (arxiv/1406.3399)
Thanks!

Questions?
Backup Slides
Named Graphs

prefix foaf: http://xmlns.com/foaf/0.1/
prefix ex: http://example.name/

ex:mygraph { ( ex:alice , foaf:knows , ex:bob ) }  
( ex:alice , foaf:name , "Alice" )  
( ex:bob , foaf:name , "Bob" )  
( ex:bob , foaf:age , 23 )  

( ex:mygraph , ex:certainty , 0.8 )
Querying Named Graphs

SELECT ?c
WHERE {
    GRAPH ?g {
        ex:alice foaf:knows ex:bob
    }
    ?g ex:certainty ?c
}
Querying Named Graphs

```
SELECT ?c
WHERE {
  GRAPH ?g {
    ex:alice foaf:knows ex:bob
    { SELECT ( COUNT(*) AS ?cnt )
      WHERE { ?s ?p ?o } }
  }
  FILTER ( ?cnt = 1 )
}
?g ex:certainty ?c
```
SELECT ?c
WHERE {
    "<<ex:alice foaf:knows ex:bob>>" ex:certainty ?c
}

SELECT ?c
WHERE {
    BIND ( "<<ex:alice foaf:knows ex:bob>>" AS ?s )
    ?s ex:certainty ?c
}
Definition: Property Graph

A *Property Graph* is a tuple $G=(V,E,src,tgt,lbl,\Phi)$ s.t.

- $(V,E,src,tgt,lbl)$ is a labeled multigraph, i.e.,
  - $V$ and $E$ are vertices and edges, respectively,
  - $src: E \rightarrow V$, $tgt: E \rightarrow V$, and $tgt: E \rightarrow S$; and
- $\Phi$ is a function that maps every vertex and edge to a finite set of pairs $p = (k,v)$ such that $k$ is a string and $v$ is a value from the domain of some (programming language specific) datatype.
Definition: Edge Uniqueness

A Property Graph $G = (V,E,src,tgt,lbl,\Phi)$ is edge-unique if it does not contain a pair of distinct edges $e$ and $e'$ such that $src(e) = src(e')$, $tgt(e) = tgt(e')$, and $lbl(e) = lbl(e')$. 

![Diagram showing edge uniqueness with certainty values of 0.8 and 0.5]
Definition: Property Uniqueness

Property Graph $G = (V,E,src,tgt,lbl,\Phi)$ is property-unique if, for each vertex or edge $x$, $k \neq k'$ for all pairs of distinct properties $(k,v)$ and $(k',v')$ in $\Phi(x)$.

• Implementations usually assume (resp. enforce) property uniqueness

• For some RDF* graphs, the transformations result in a Property Graph that is not property-unique.