Outline

Navigation and Patterns

Rule-based languages

Moving to RDF

Outline

Navigation and Patterns

Rule-based languages

Path Queries for RDF

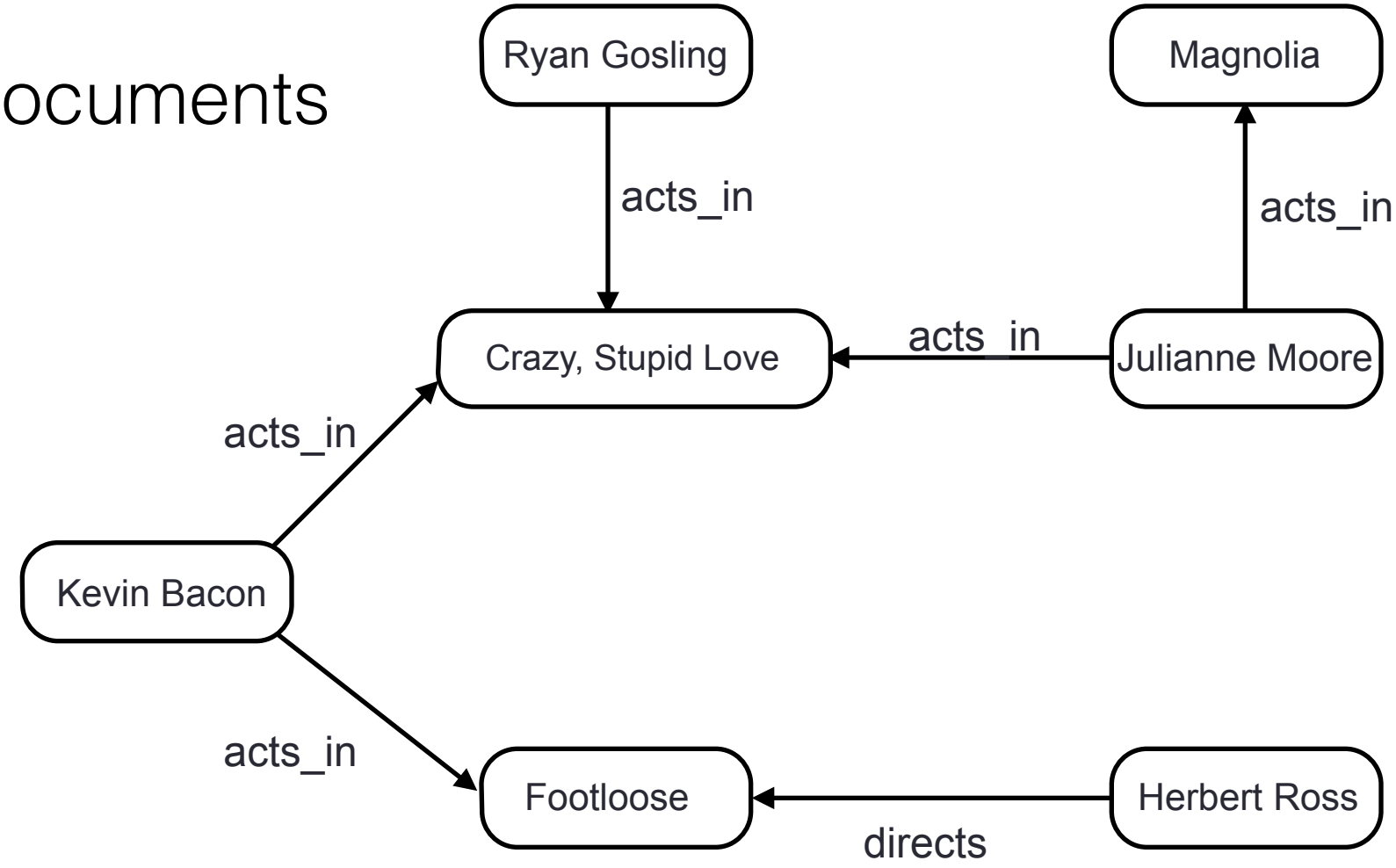
Moving to RDF

Triple Algebra

Practical Concerns

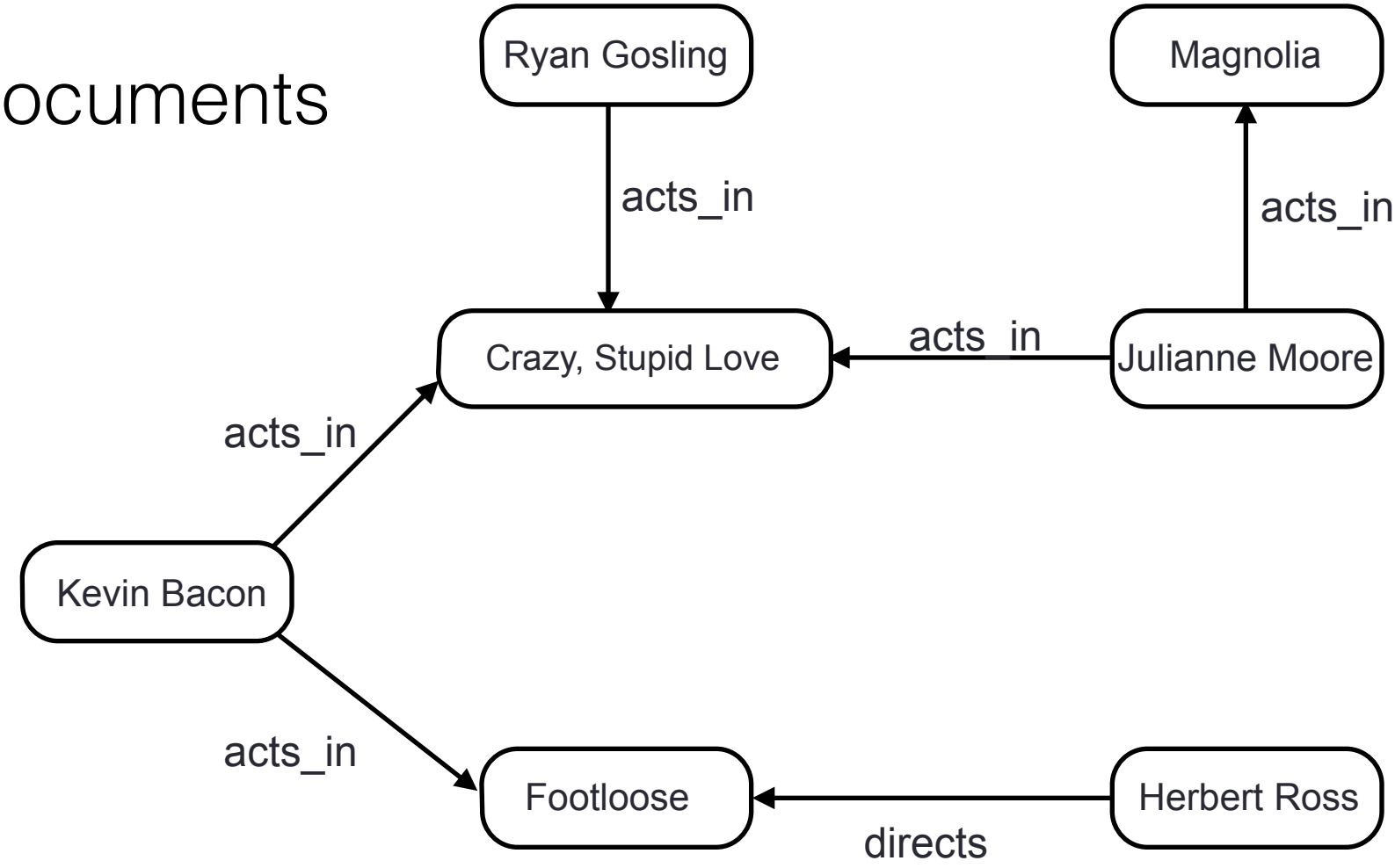
Lifting graph languages to support RDF graphs

RDF documents



Subject	Predicate	Object
Bacon	acts_in	Footloose
Ross	directs	Footloose
Moore	acts_in	Magnolia

RDF documents



Subject	Predicate	Object
Bacon	acts_in	Footloose
Ross	directs	Footloose
Moore	acts_in	Magnolia

big difference v/s graphs:
labels are not fixed
Predicates store information

RDF documents

Subject	Predicate	Object
Bacon	acts_in	Footloose
Ross	directs	Footloose
Moore	acts_in	Magnolia

Current navigation approaches treat RDF documents as graphs

acts_in	
Bacon	Footloose
More	Magnolia

directs	
Ross	Footloose

Property Paths

Essentially adding 2RPQs to SPARQL (query language for RDF)

Property Paths

Essentially adding 2RPQs to SPARQL (query language for RDF)

To evaluate 2RPQs in RDF we treat them as graphs,
the predicates becomes the labels

Subject	Predicate	Object
Bacon	acts_in	Footloose
Ross	directs	Footloose
Moore	acts_in	Magnolia

acts_in	
Bacon	Footloose
More	Magnolia

directs	
Ross	Footloose

Property Paths

Essentially adding 2RPQs to SPARQL (query language for RDF)

To evaluate 2RPQs in RDF we treat them as graphs,
the predicates becomes the labels

Good thing: [Kostylev, R., Romero, Vrgoc 15]

Evaluation/Containment results from graphs
transfer to Property Paths

Problem:
SPARQL + Property Paths requires a mixed strategy

Problem:
SPARQL + Property Paths requires a mixed strategy

SPARLQ logically treats RDF documents as triples
Property Paths logically treats RDF documents as graph

SPARLQ logically treats RDF documents as triples ,
outputs either mappings or RDF

Property Paths logically treats RDF documents as graph,
the output are pairs of nodes

SPARLQ logically treats RDF documents as triples ,
outputs either mappings or RDF

Property Paths logically treats RDF documents as graph,
the output are pairs of nodes

Even worse than lacking algebraic closure!

Very difficult to implement (requires different engines,
not clear how to optimise…)

Need a language

- Capable of expressing basic navigation (at least path queries)
- Closed Input/output:
  Evaluated over RDF,
  Producing RDF as input
- Algebraically closed

Need a language

- Capable of expressing basic navigation (at least path queries)
- Input/output closed:
  Evaluated over RDF,
  Producing RDF as input
- Algebraically closed

We already learned: need transitive closure over patterns

Need a language

- Capable of expressing basic navigation (at least path queries)
- Input/output closed:
  Evaluated over RDF,
  Producing RDF as input
- Algebraically closed

We already learned: need transitive closure over patterns

Problem is, patterns are no longer binary!

Outline

Navigation and Patterns

Rule-based languages

Path Queries for RDF

Moving to RDF

Triple Algebra

Practical Concerns

Defining the P^{+} operator over tertiary relations

For the binary case it's just the composition of relations

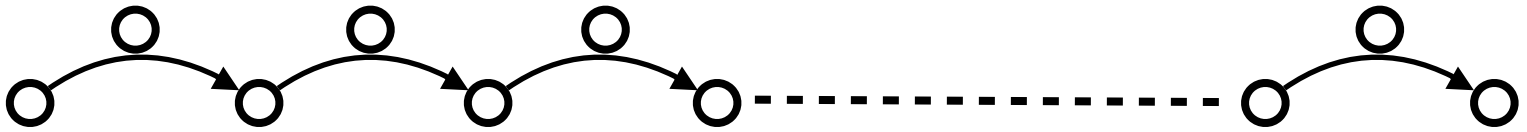
Defining the P^+ operator over tertiary relations

For the binary case it's just the composition of relations

But for tertiary relations we have several possibilities

How to define the P^+ operator?

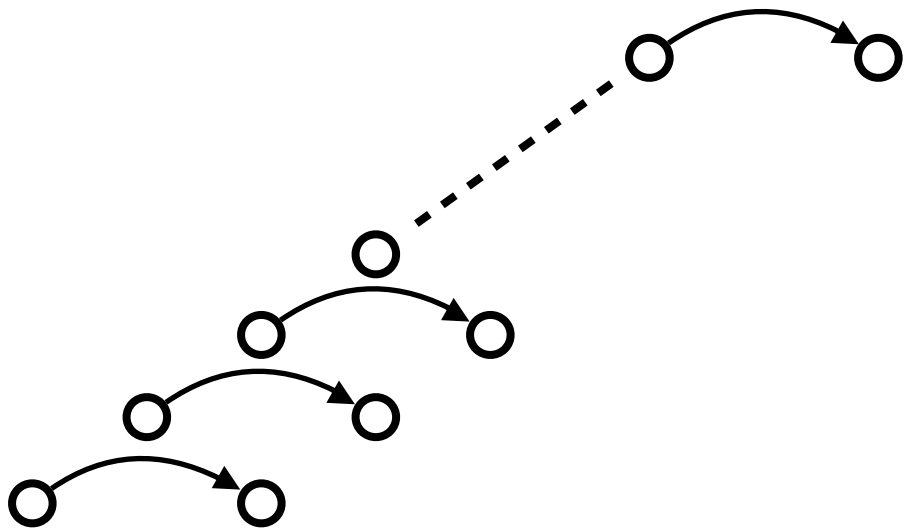
How to define the P^+ operator?



Classical graph reachability

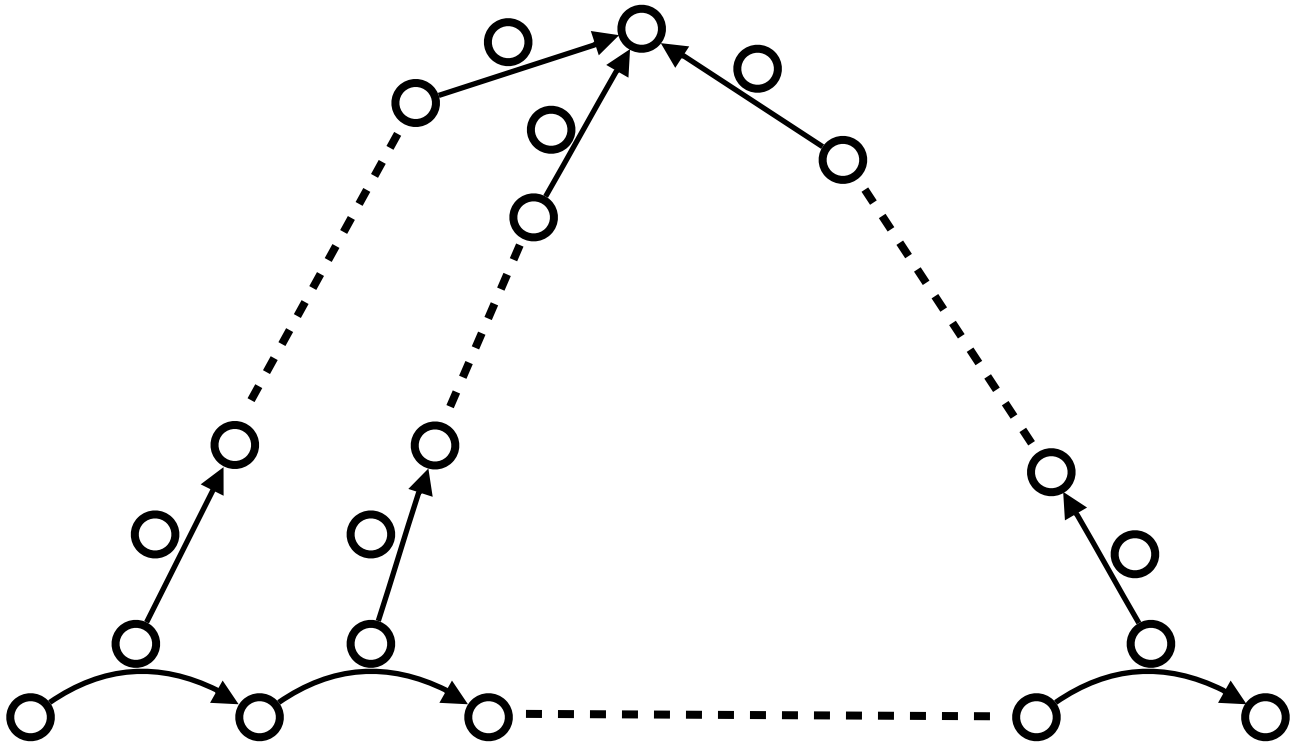
(middle element of triples represent labels, or properties)

How to define the P^+ operator?



Another possibility:
Use the middle element as start of next triple

How to define the P^+ operator?

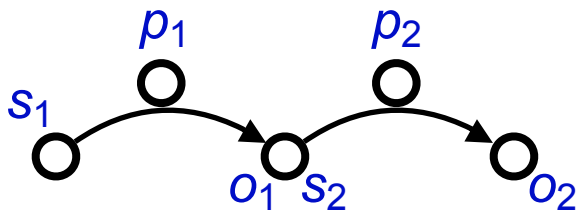


Combining different ways of reachability

How to define the P^+ operator?

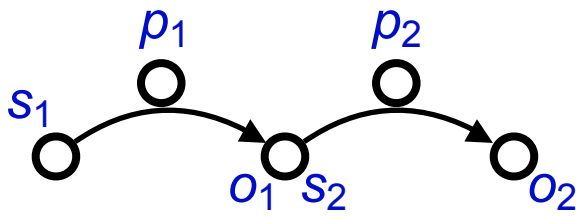
Main Idea: don't choose, just take all possibilities

Reachability over tertiary relations



Think of this operation:

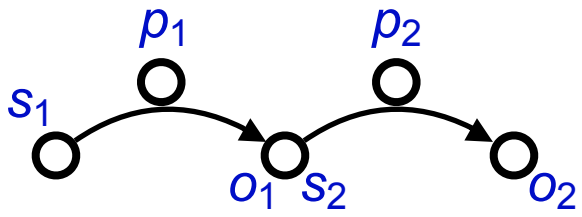
Reachability over tertiary relations



Think of this operation:

$$T(s_1, p_2, o_2) \leftarrow T(s_1, p_1, o_1), T(s_2, p_2, o_2) \;\; o_1 = s_2$$

Reachability over tertiary relations

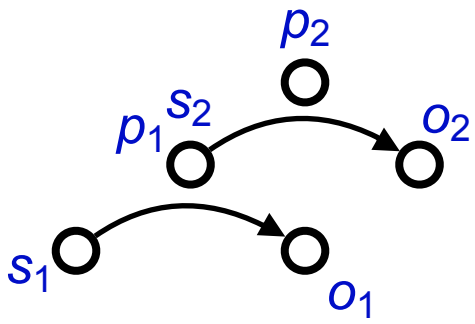


Think of this operation:

$$T(s_1, p_2, o_2) \leftarrow T(s_1, p_1, o_1), T(s_2, p_2, o_2) \;\; o_1 = s_2$$

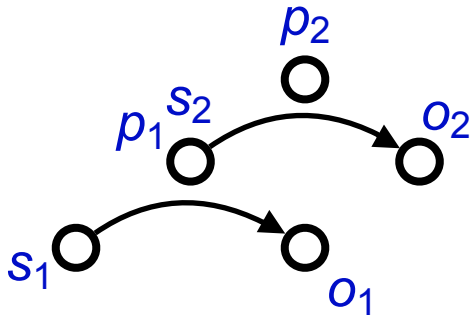
$$T \underset{o_1 = s_2}{\overset{(s_1, p_2, o_2)}{\bowtie}} T$$

Reachability over tertiary relations



Think of this operation:

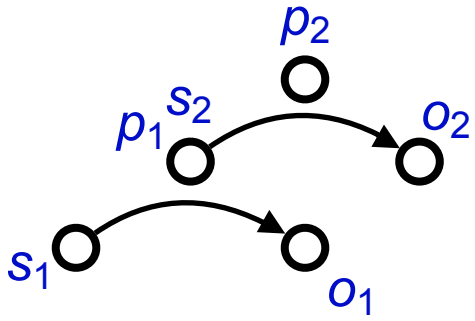
Reachability over tertiary relations



Think of this operation:

$$T(s_1, p_2, o_2) \leftarrow T(s_1, p_1, o_1),\ T(s_2, p_2, o_2),\ p_1 = s_2$$

Reachability over tertiary relations

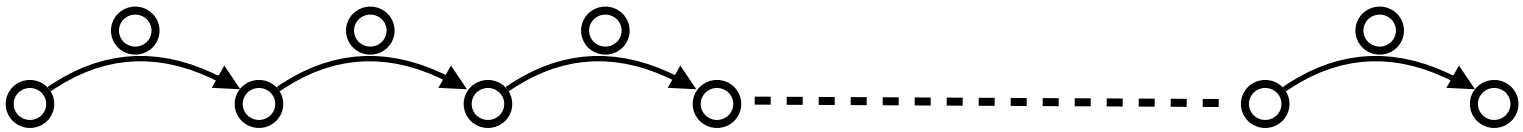


Think of this operation:

$$T(s_1, p_2, o_2) \leftarrow T(s_1, p_1, o_1),\ T(s_2, p_2, o_2),\ p_1 = s_2$$

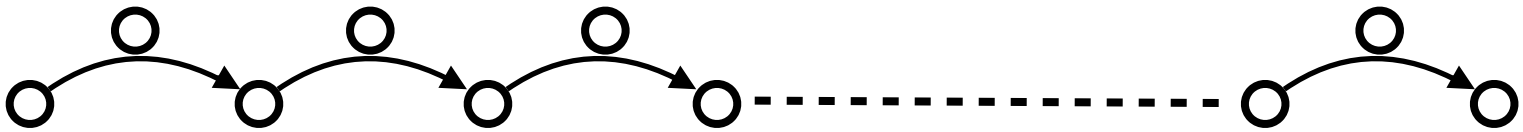
$$T \overset{(s_1, p_2, o_2)}{\underset{p_1 = s_2}{\bowtie}} T$$

Reachability over tertiary relations



Classical graph reachability:

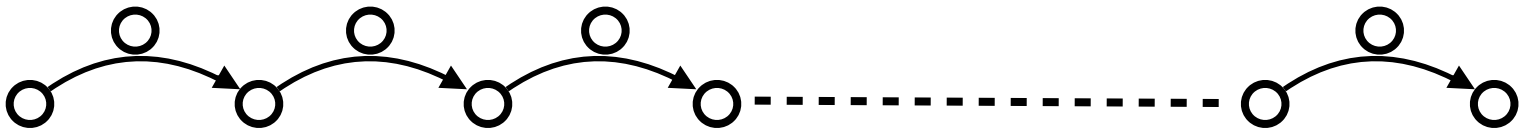
Reachability over tertiary relations



Classical graph reachability:

$$\left(T \overset{(s_1, p_2, o_2)}{\underset{o_1 = s_2}{\bowtie}} \right)^+$$

Reachability over tertiary relations

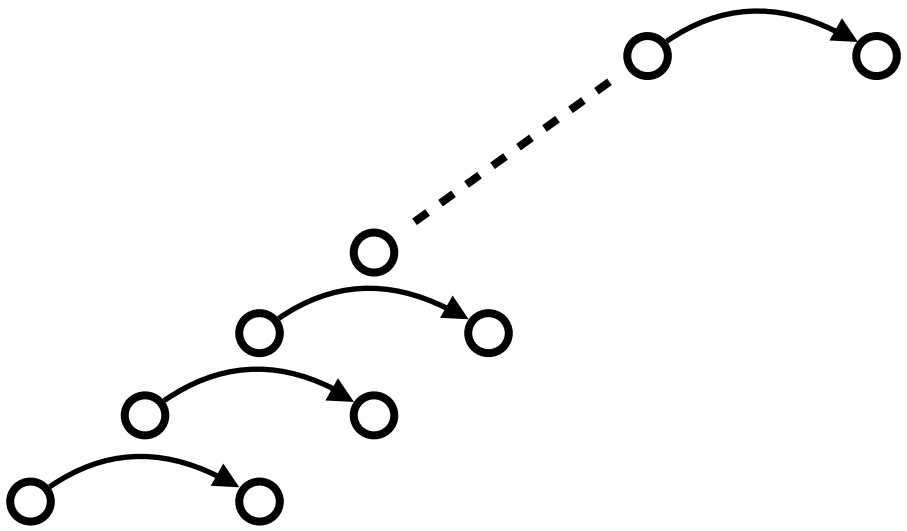


Classical graph reachability:

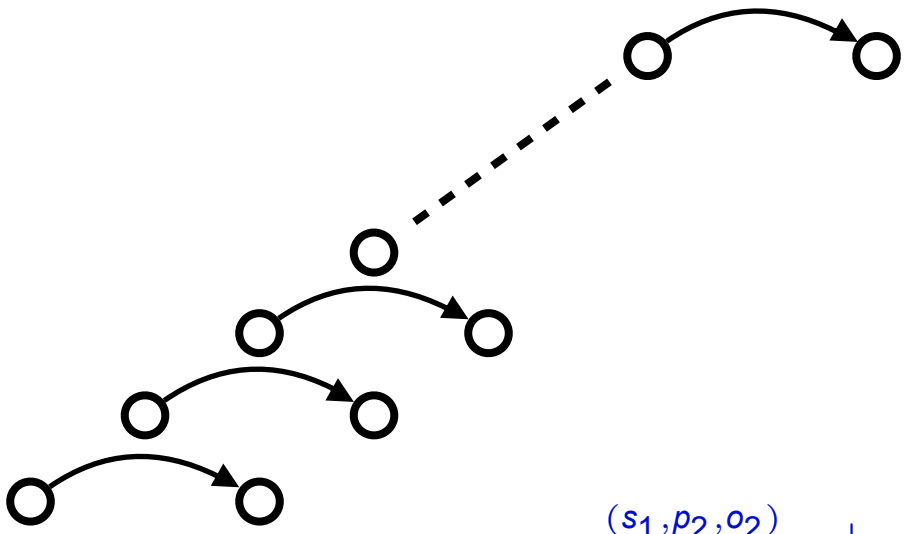
$$\left(T \underset{o_1 = s_2}{\overset{(s_1, p_2, o_2)}{\bowtie}} \right)^+$$

$$= T \cup T \underset{o_1 = s_2}{\overset{(s_1, p_2, o_2)}{\bowtie}} T \cup \left(T \underset{o_1 = s_2}{\overset{(s_1, p_2, o_2)}{\bowtie}} T \right) \underset{o_1 = s_2}{\overset{(s_1, p_2, o_2)}{\bowtie}} T \cup \cdots$$

Reachability over tertiary relations

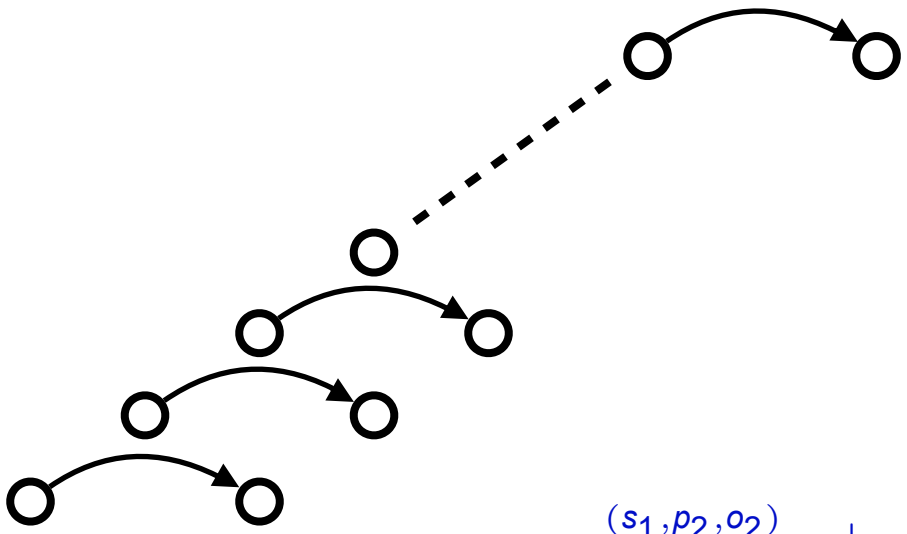


Reachability over tertiary relations



$$\left(T \overset{(s_1, p_2, o_2)}{\underset{p_1 = s_2}{\bowtie}} \right)^+$$

Reachability over tertiary relations



$$\left(T \underset{p_1 = s_2}{\overset{(s_1, p_2, o_2)}{\bowtie}} \right)^+$$

$$= T \cup T \underset{p_1 = s_2}{\overset{(s_1, p_2, o_2)}{\bowtie}} T \cup \left(T \underset{p_1 = s_2}{\overset{(s_1, p_2, o_2)}{\bowtie}} T\right) \underset{p_1 = s_2}{\overset{(s_1, p_2, o_2)}{\bowtie}} T \cup \cdots$$

Triple Algebra (TriAL)

Triple Algebra (TriAL)

T is a TriAL expression
(T represents the whole set of triples)

Triple Algebra (TriAL)

T is a TriAL expression
(T represents the whole set of triples)

if R and R' are TriAL expressions, then

$R \cup R'$ and $R - R'$ are TriAL expressions

Triple Algebra (TriAL)

T is a TriAL expression
(T represents the whole set of triples)

if R and R' are TriAL expressions, then

$R \cup R'$ and $R - R'$ are TriAL expressions

$R \overset{(x,y,z)}{\underset{\phi}{\bowtie}} R'$ is a TriAL expression, for

Triple Algebra (TriAL)

T is a TriAL expression
(T represents the whole set of triples)

if R and R' are TriAL expressions, then

$R \cup R'$ and $R - R'$ are TriAL expressions

$R \overset{(x,y,z)}{\underset{\phi}{\bowtie}} R'$ is a TriAL expression, for

$x, y, z \in \{s_1, s_2, p_1, p_2, o_2, o_2\}$

ϕ equalities over $\{s_1, s_2, p_1, p_2, o_2, o_2\}$

Positive TriAL

T is a TriAL expression
(T represents the whole set of triples)

if R and R' are TriAL expressions, then

$R \cup R'$ and ~~$R - R'$~~ are TriAL expressions

$R \overset{(x,y,z)}{\underset{\phi}{\bowtie}} R'$ is a TriAL expression, for

$x, y, z \in \{s_1, s_2, p_1, p_2, o_2, o_2\}$

ϕ equalities over $\{s_1, s_2, p_1, p_2, o_2, o_2\}$

Recursive TriAL

$$(R \bowtie)^{+} = R \cup R \bowtie R \cup (R \bowtie R) \bowtie R \cup \cdots$$

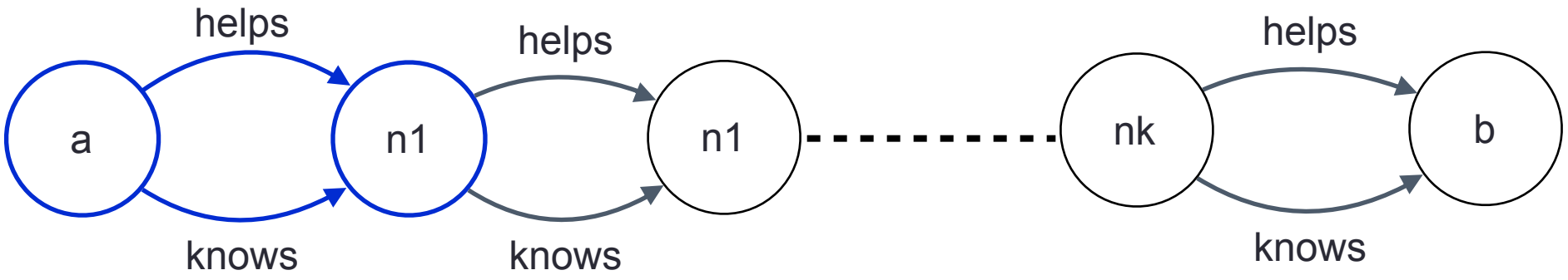
Recursive TriAL

$$(R \bowtie)^+ = R \cup R \bowtie R \cup (R \bowtie R) \bowtie R \cup \cdots$$

$$(\bowtie R)^+ = R \cup R \bowtie R \cup R \bowtie (R \bowtie R) \cup \cdots$$

(note that these two are not necessarily the same)

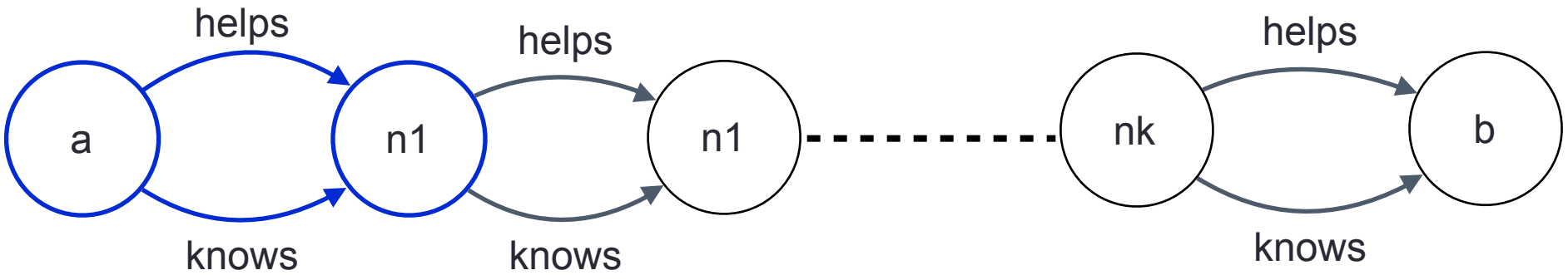
Recursive Triple Algebra (examples)



Say u is a friend of v if
u knows v and u helps v

Find all nodes connected
by a chain of friends

Recursive Triple Algebra (examples)

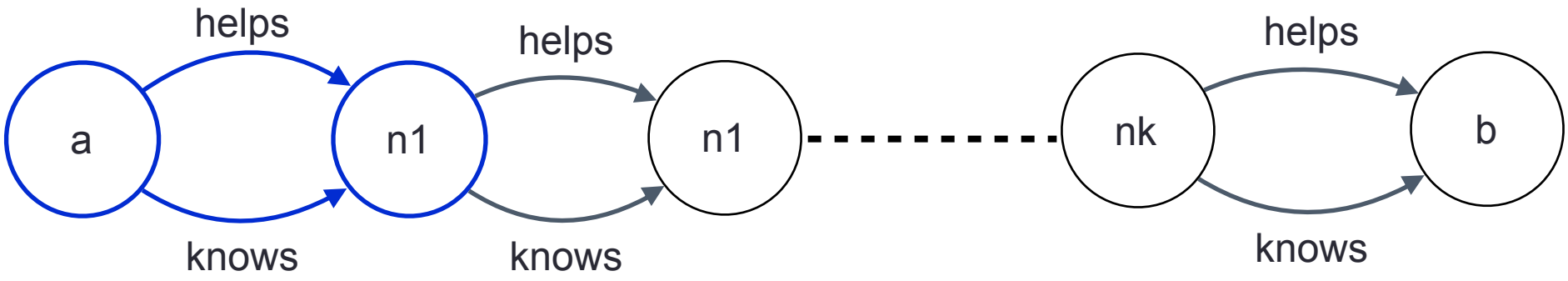


Say u is a friend of v if
u knows v and u helps v

Find all nodes connected
by a chain of friends

Friends: $T \underset{s_1=s_2,\ o_1=o_2,\ p_1=\text{helps},\ p_2=\text{knows}}{\overset{(s_1,p_1,o_1)}{\bowtie}} R'$

Recursive Triple Algebra (examples)



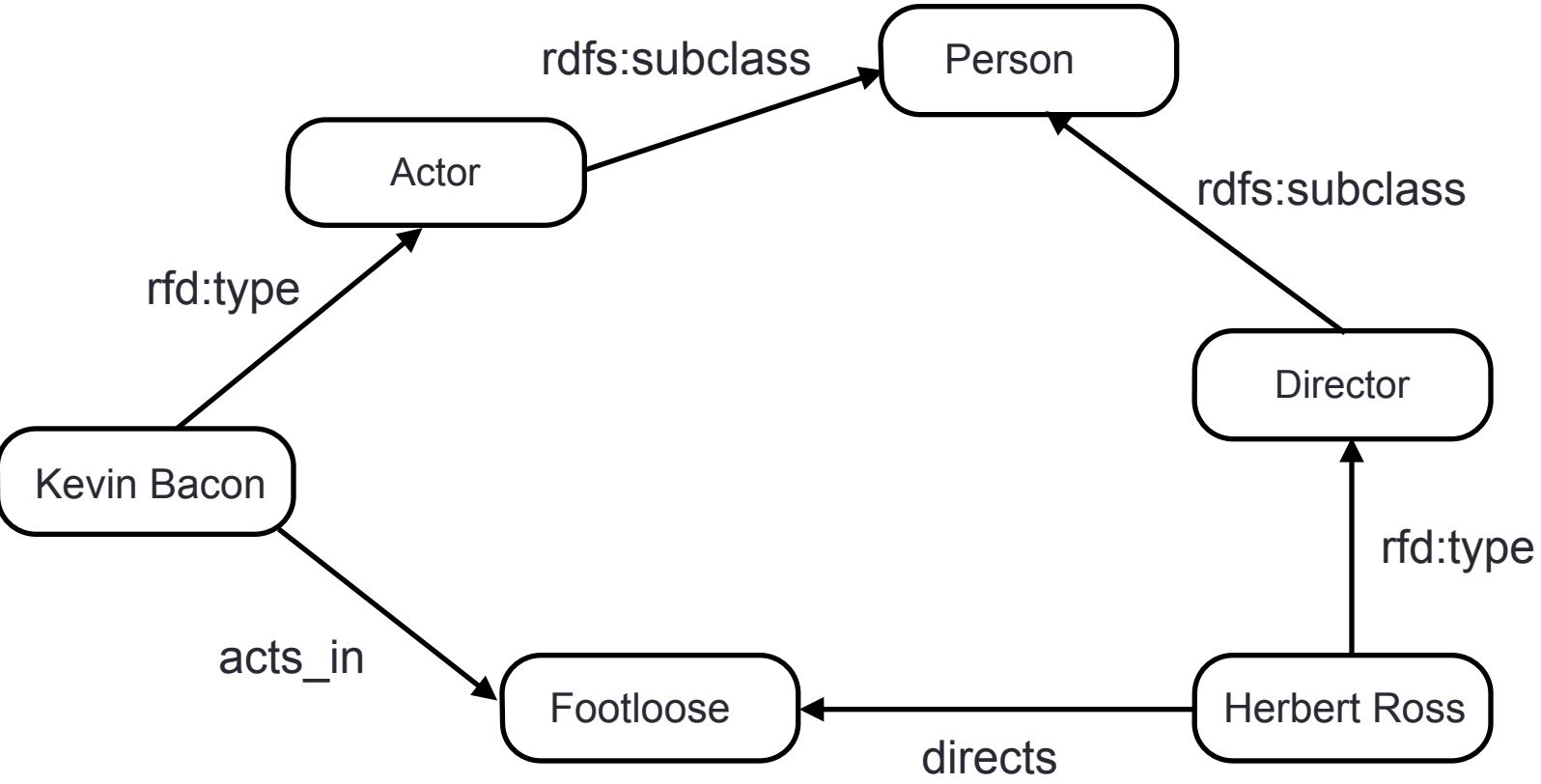
Say u is a friend of v if
u knows v and u helps v

Find all nodes connected
by a chain of friends

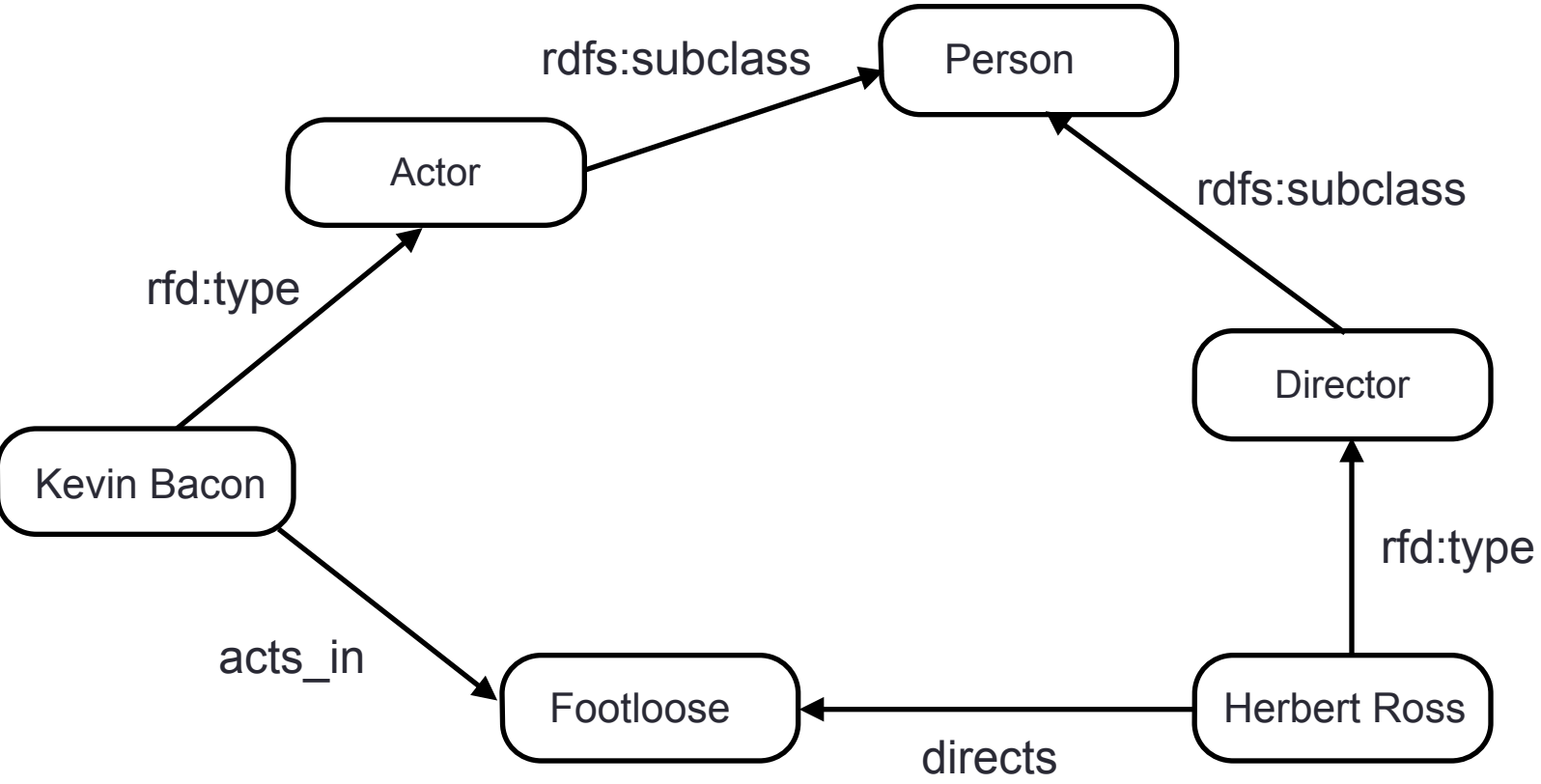
$$\text{Friends: } T \overset{(s_1, p_1, o_1)}{\underset{s_1 = s_2,\ o_1 = o_2,\ p_1 = \text{helps},\ p_2 = \text{knows}}{\bowtie}} R'$$

$$\left(\text{Friends} \overset{(s_1, p_2, o_2)}{\underset{o_1 = s_2}{\bowtie}}\right)^{+}$$

Recursive Triple Algebra (examples)



Recursive Triple Algebra (examples)



Applying transitivity of subclass:

$$\left(T \underset{o_1=s_2,\ p_1=\text{rdf:type},\ p_2=\text{rdf:subclass}}{\overset{(s_1,p_1,o_2)}{\bowtie}}\right)^{+}$$

Evaluation

Is a triple (s,p,o) in the evaluation of a recursive TriAL expression over a graph?

Evaluation

Is a triple (s,p,o) in the evaluation of a recursive TriAL expression over a graph?

Evaluation of recursive TriAL is: [Libkin, R., Vrgoc 13]

PTIME-complete
NLogSpace-complete if the expression is fixed

(almost the same than graph path queries)

Recursive Triple Algebra as Query Language

TriAL expressions are algebraically closed

but it is only a navigational primitive,
does not subsume CQs (graph patterns)

Think of them as the triplestore equivalent of path queries

Recursive Triple Algebra as Primitive

Can add TriAL expressions to CQs (graph patterns).
but language not algebraically closed anymore.

Recursive Triple Algebra as Primitive

Can add TriAL expressions to CQs (graph patterns).
but language not algebraically closed anymore.

Other option: Triplestore equivalent of regular queries?

Regular Queries using TriAL

$$S(z_1, z_2, z_3) \leftarrow R_1(u_1, v_1, w_1), \ \ldots, \ R_n(u_n, v_n, w_n)$$

Each R_i is either a predicate, a label,
or expressions $(P \bowtie)^+$ or $(\bowtie P)^+$,
for a predicate P

Same as before, queries are sets of non-recursive rules

Regular Queries using TriAL

$$S(z_1, z_2, z_3) \leftarrow R_1(u_1, v_1, w_1), \ \ldots, \ R_n(u_n, v_n, w_n)$$

Each R_i is either a predicate, a label,
or expressions $(P \bowtie)^+$ or $(\bowtie P)^+$,
for a predicate P

Evaluation remains NP-complete
(NLogSpace-c if program is fixed)

Containment?

Moving to higher arities

$$S(\bar{z}) \leftarrow R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$$

Each R_i is either a predicate, a label,
or expressions $(P \bowtie)^+$ or $(\bowtie P)^+$,
for a predicate P

Alternative for systems to implement
navigational patterns using a single engine

Recap

Definition of navigational primitives for triples is not immediate

Recap

Definition of navigational primitives for triples is not immediate

One option: Do “kleene closure” of joins:

$$(R \bowtie)^+ = R \cup R \bowtie R \cup (R \bowtie R) \bowtie R \cup \cdots$$

$$(\bowtie R)^+ = R \cup R \bowtie R \cup R \bowtie (R \bowtie R) \cup \cdots$$

Recap

Definition of navigational primitives for triples is not immediate

One option: Do “kleene closure” of joins:

$$(R \bowtie)^+ = R \cup R \bowtie R \cup (R \bowtie R) \bowtie R \cup \cdots$$

$$(\bowtie R)^+ = R \cup R \bowtie R \cup R \bowtie (R \bowtie R) \cup \cdots$$

Advantages: More expressive power,
Can compute navigation using Triplestore engine

Outline

Navigation and Patterns

Rule-based languages

Path Queries for RDF

Moving to RDF

Triple Algebra

Practical Concerns

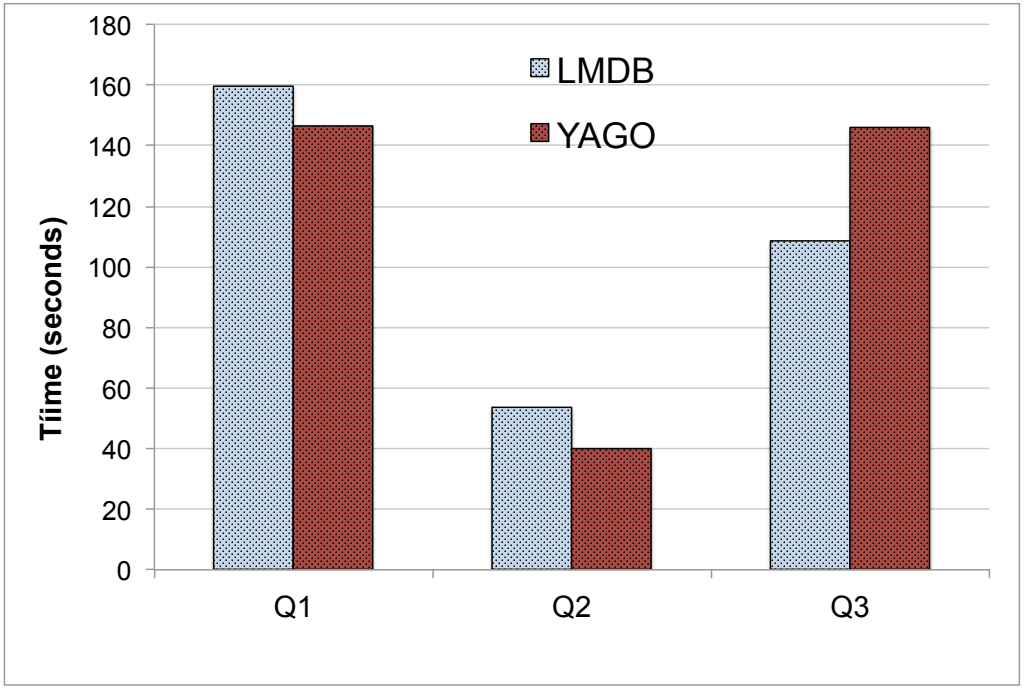
Does this work in practice at all?

Implemented a (superset) of this language, on top of SPARQL
- Recursive SPARQL

Needed to add comparison with constants
(very important in practice)

Datasets:

1. Linked Movie Database
2. Yago (only movie facts)



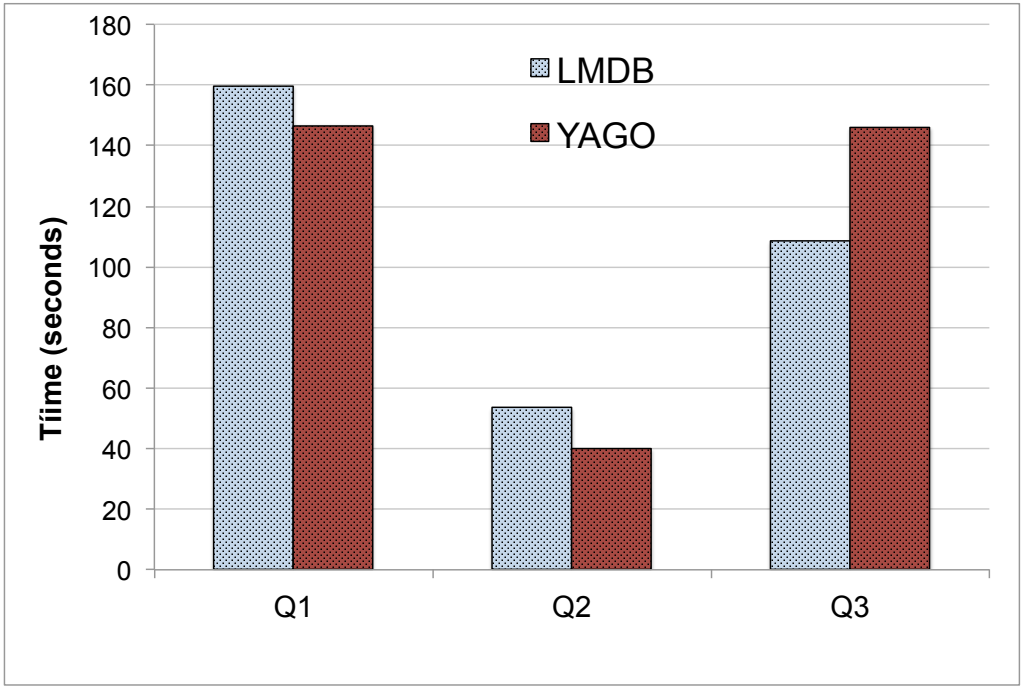
[R., Soto, Vrgoc 15]

Queries:

1. Actors connected to Kevin Bacon via $(\text{acted\_in} \mid \text{acted\_in}^{-})^{+}$
2. Actors connected to Kevin Bacon through movies with same director.
3. Actors connected to Bacon via movies in which the director is also an actor (like Clint Eastwood).

Datasets:

1. Linked Movie Database
2. Yago (only movie facts)



[R., Soto, Vrgoc 15]

Can express as a RQ or a recursive TriAL exp.

Queries:

1. Actors connected to Kevin Bacon via $(acted\_in \mid acted\_in^{-})^{+}$
2. Actors connected to Kevin Bacon through movies with same director.
3. Actors connected to Bacon via movies in which the director is also an actor (like Clint Eastwood).

Comparison with SPARQL Path Queries (Property Paths)

Compare our alternative with two RDF/SPARQL engines:

- Virtuoso
- Jena

Comparison with SPARQL Path Queries (Property Paths)

Compare our alternative with two RDF/SPARQL engines:

- Virtuoso
- Jena

none of them can compute all actors
connected to Kevin Bacon (under default settings).

We do it in 160 seconds

Outline

Navigation and Patterns

Rule-based languages

Moving to RDF

Conclusions and Main Takeaways

Main Takeaways

Main Takeaways

Graph DBs: Pattern matching + Navigation

Main Takeaways

Graph DBs: Pattern matching + Navigation

Slowly showing up in industry.

Need to think about algebraically closed languages

Main Takeaways

Graph DBs: Pattern matching + Navigation

Slowly showing up in industry.
Need to think about algebraically closed languages

Already have alternatives, but need to keep working!

Specially for RDF

Challenges and open problems

Challenges and open problems

User friendly Datalog-based query languages
Translation from Datalog to lower level algebra for systems

Challenges and open problems

User friendly Datalog-based query languages
Translation from Datalog to lower level algebra for systems

Queries returning paths.

- any path?
- shortest path?
- all paths?

Challenges and open problems

User friendly Datalog-based query languages
Translation from Datalog to lower level algebra for systems

Queries returning paths.

- any path?
- shortest path?
- all paths?

So far only navigation.

What about comparing values?
Arithmetic?