The Expressive Power of SPARQL

Renzo Angles and Claudio Gutierrez

Computer Science Department
Universidad de Chile

Motivations

- Current definition of SPARQL semantics is non-standard and unnecesarily complex

Motivations

- Current definition of SPARQL semantics is non-standard and unnecesarily complex

This paper:

SPARQL has a simple compositional semantics
(except very rare corner cases).

Motivations

- Current definition of SPARQL semantics is non-standard and unnecesarily complex

This paper:

SPARQL has a simple compositional semantics
(except very rare corner cases).

- What is the expressive power of SPARQL?
  For example, how does it compare to SQL?

Motivations

- Current definition of SPARQL semantics is non-standard and unnecesarily complex

This paper:

SPARQL has a simple compositional semantics
(except very rare corner cases).

- What is the expressive power of SPARQL?
  For example, how does it compare to SQL?

This paper:

SPARQL is equivalent in expressive power to Relational Algebra

Outline

- Overview of current definition of SPARQL semantics

Outline

- Overview of current definition of SPARQL semantics
- SPARQL is equivalent to SPARQL-S (a version of SPARQL with safe filters)

Outline

- Overview of current definition of SPARQL semantics
- SPARQL is equivalent to SPARQL-S (a version of SPARQL with safe filters)
- SPARQL-S is equivalent to SPARQL-C, a version of SPARQL with compositional semantics.

Outline

- Overview of current definition of SPARQL semantics
- SPARQL is equivalent to SPARQL-S (a version of SPARQL with safe filters)
- SPARQL-S is equivalent to SPARQL-C, a version of SPARQL with compositional semantics.
- SPARQL-C is equivalent to Relational Algebra

Expressive power of SPARQL : Tour

SPARQL
W3C Syntax and Semantics

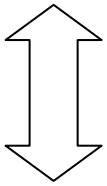
Expressive power of SPARQL : Tour

SPARQL
W3C Syntax and Semantics

SPARQL-S
Only safe-filter patterns

Expressive power of SPARQL : Tour

SPARQL
W3C Syntax and Semantics



SPARQL-S
Only safe-filter patterns

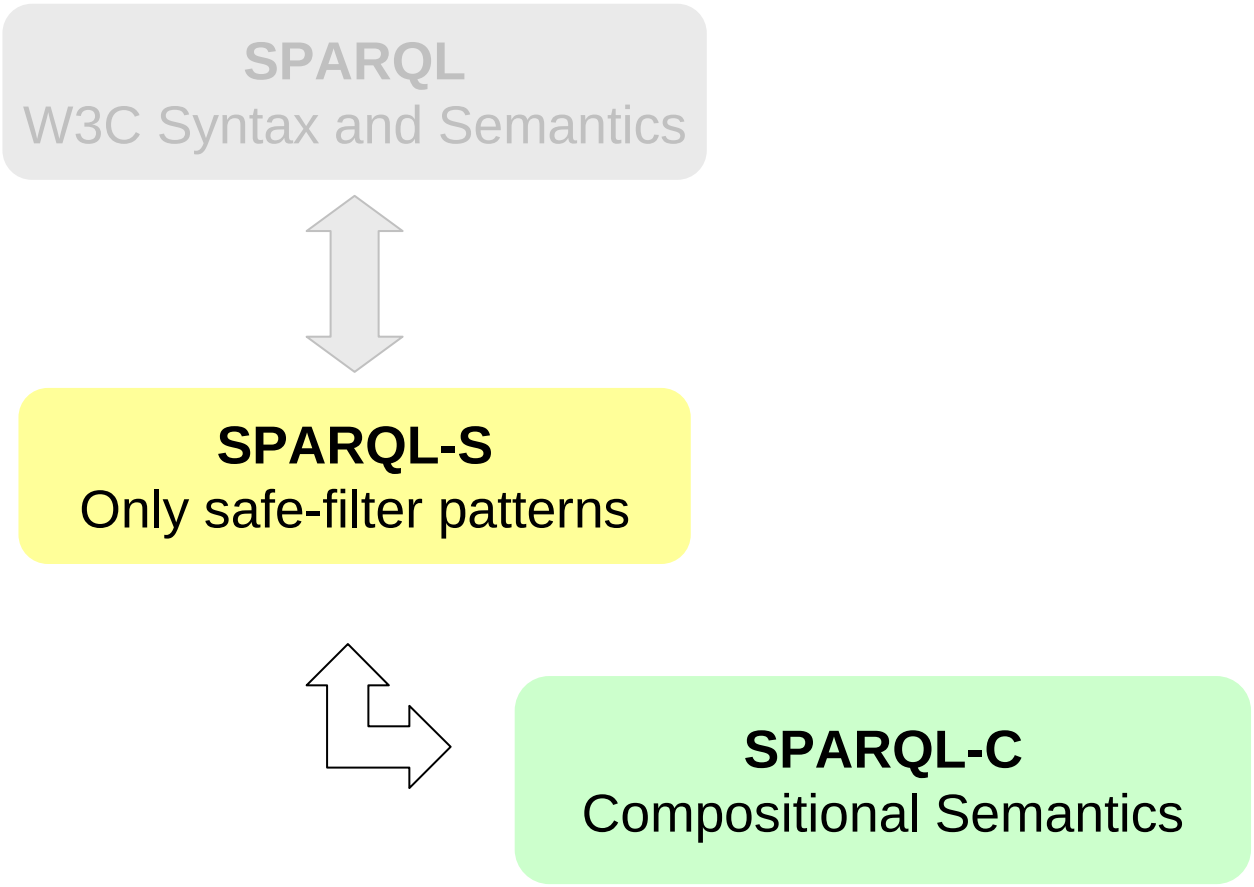
Expressive power of SPARQL : Tour

SPARQL
W3C Syntax and Semantics

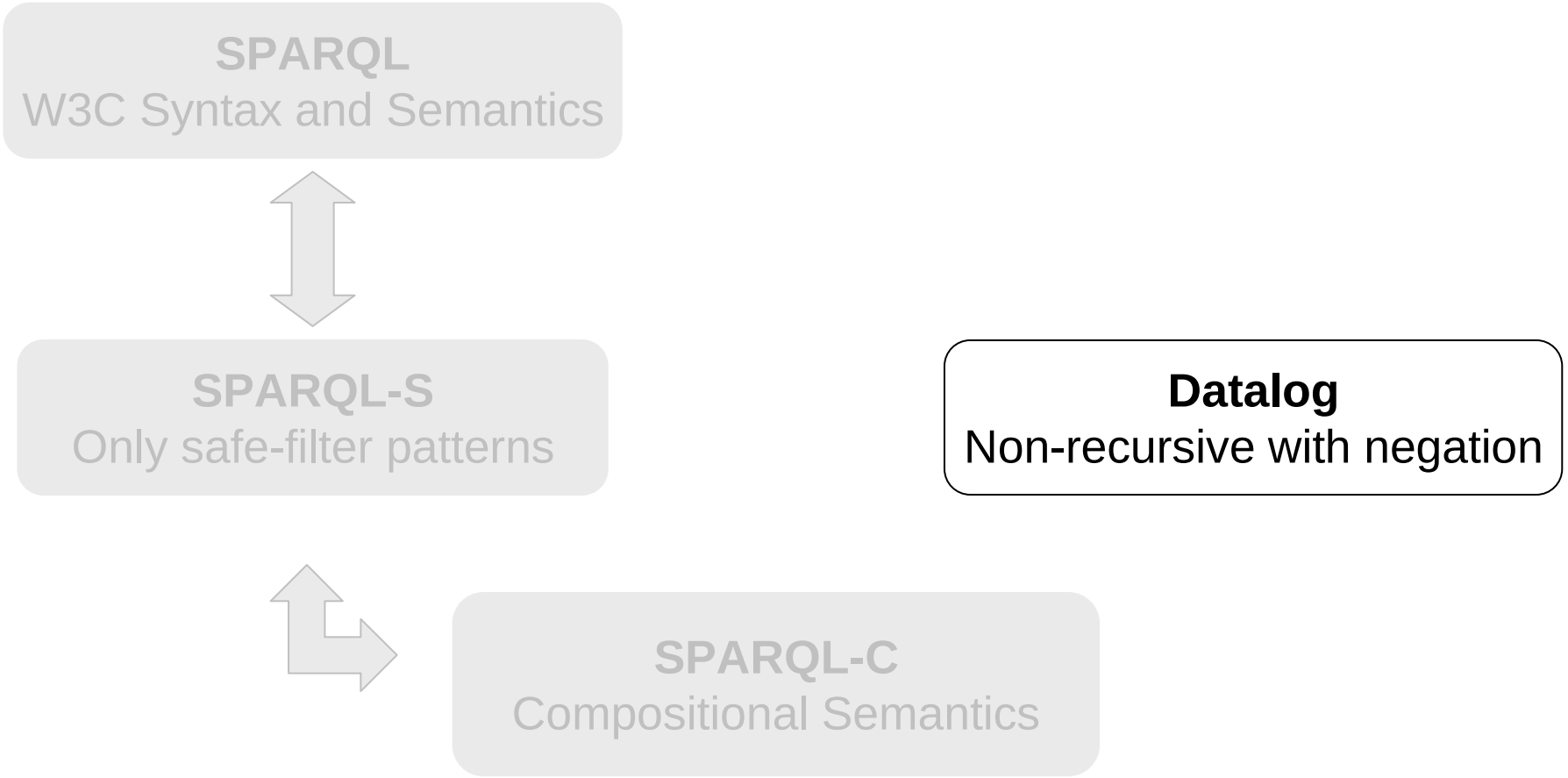
SPARQL-S
Only safe-filter patterns

SPARQL-C
Compositional Semantics

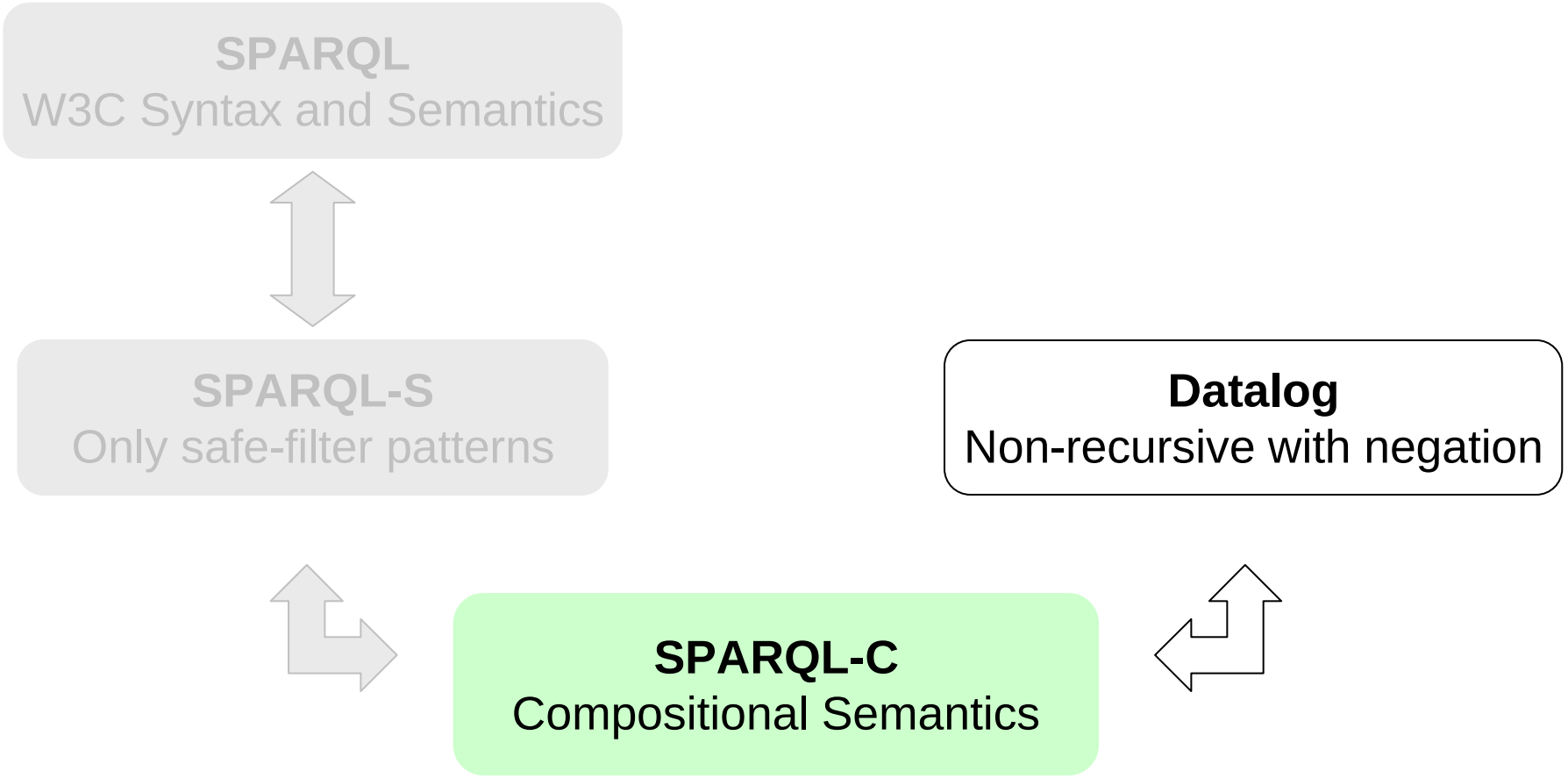
Expressive power of SPARQL : Tour



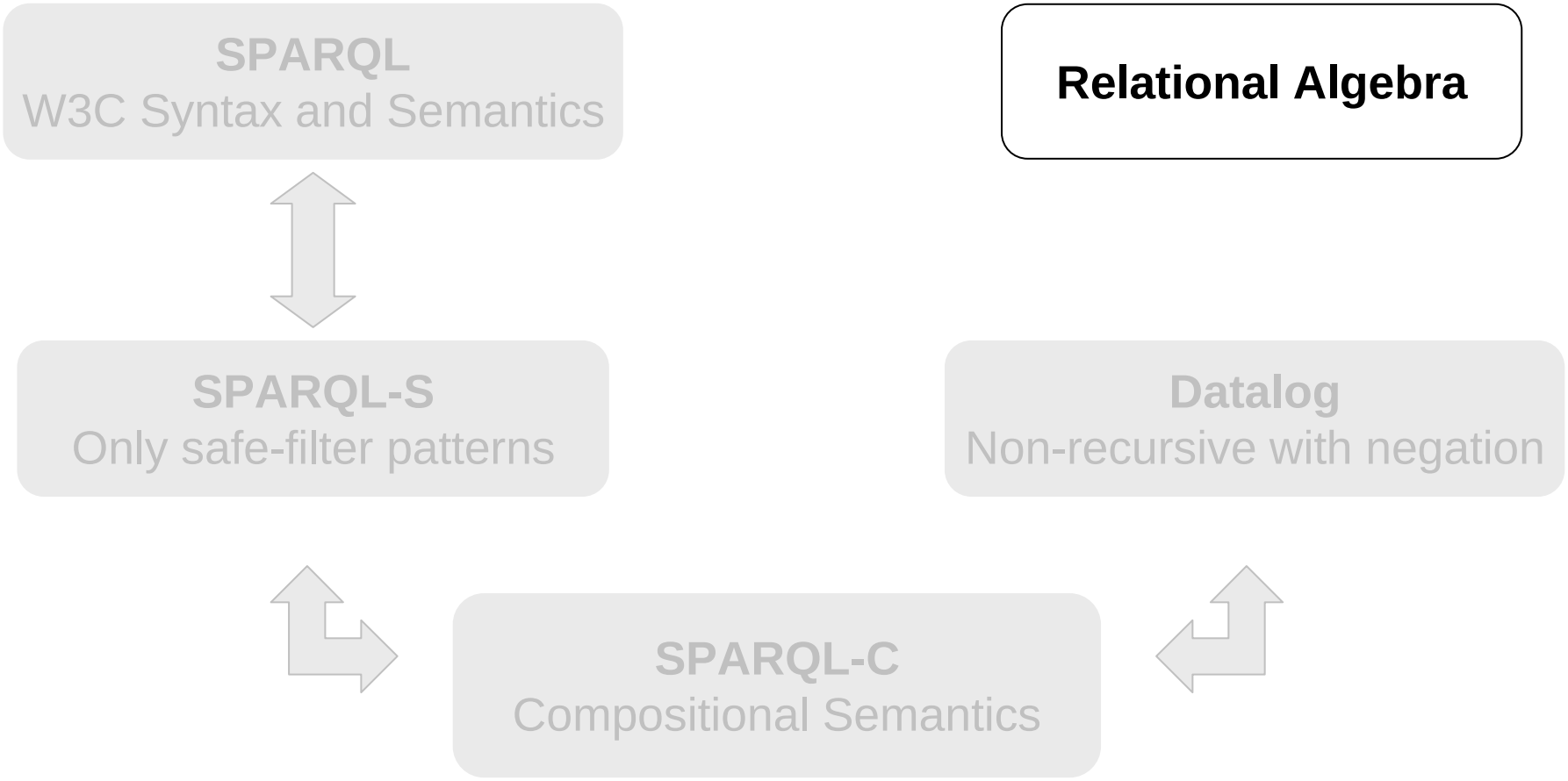
Expressive power of SPARQL : Tour



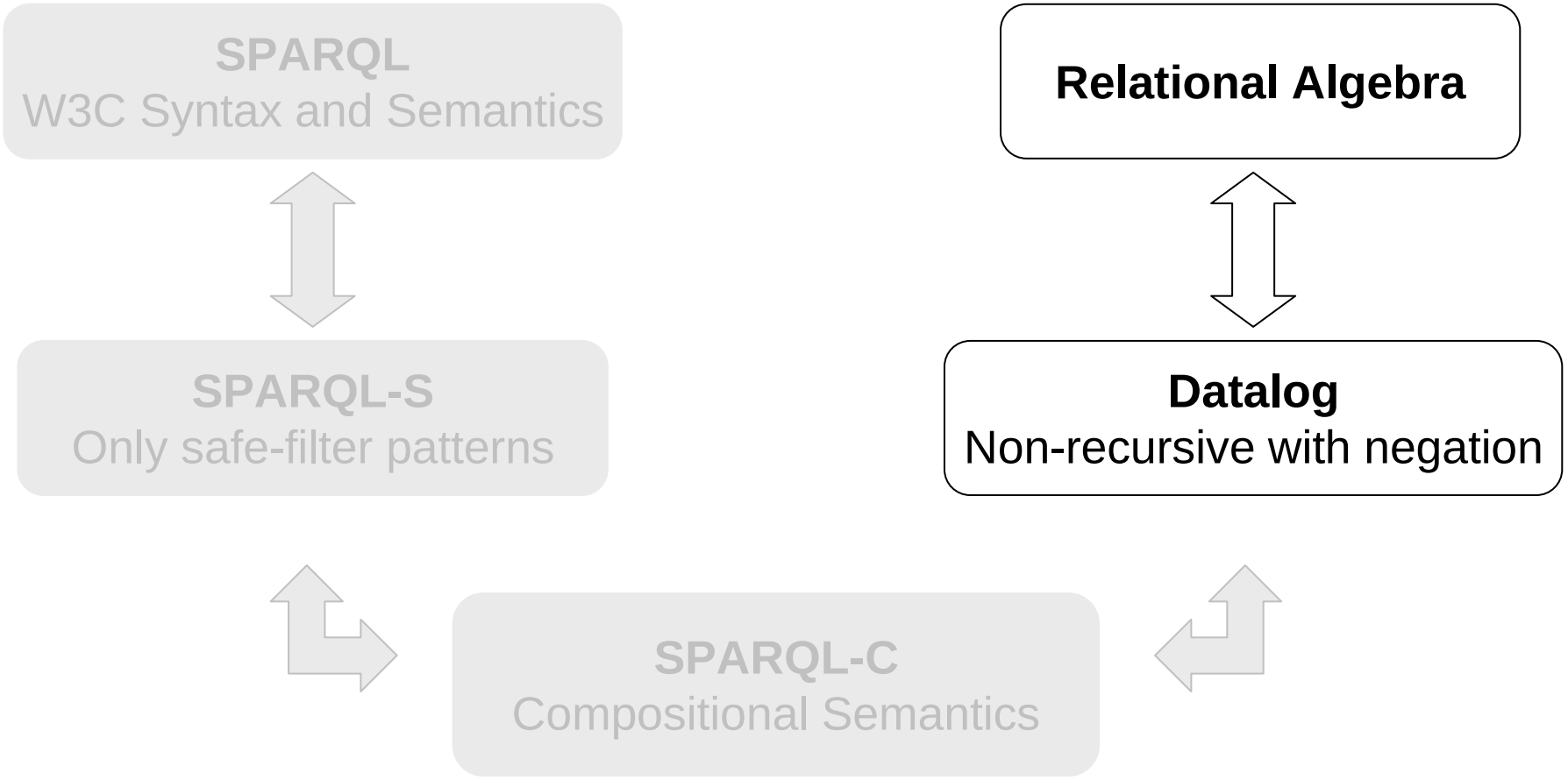
Expressive power of SPARQL : Tour



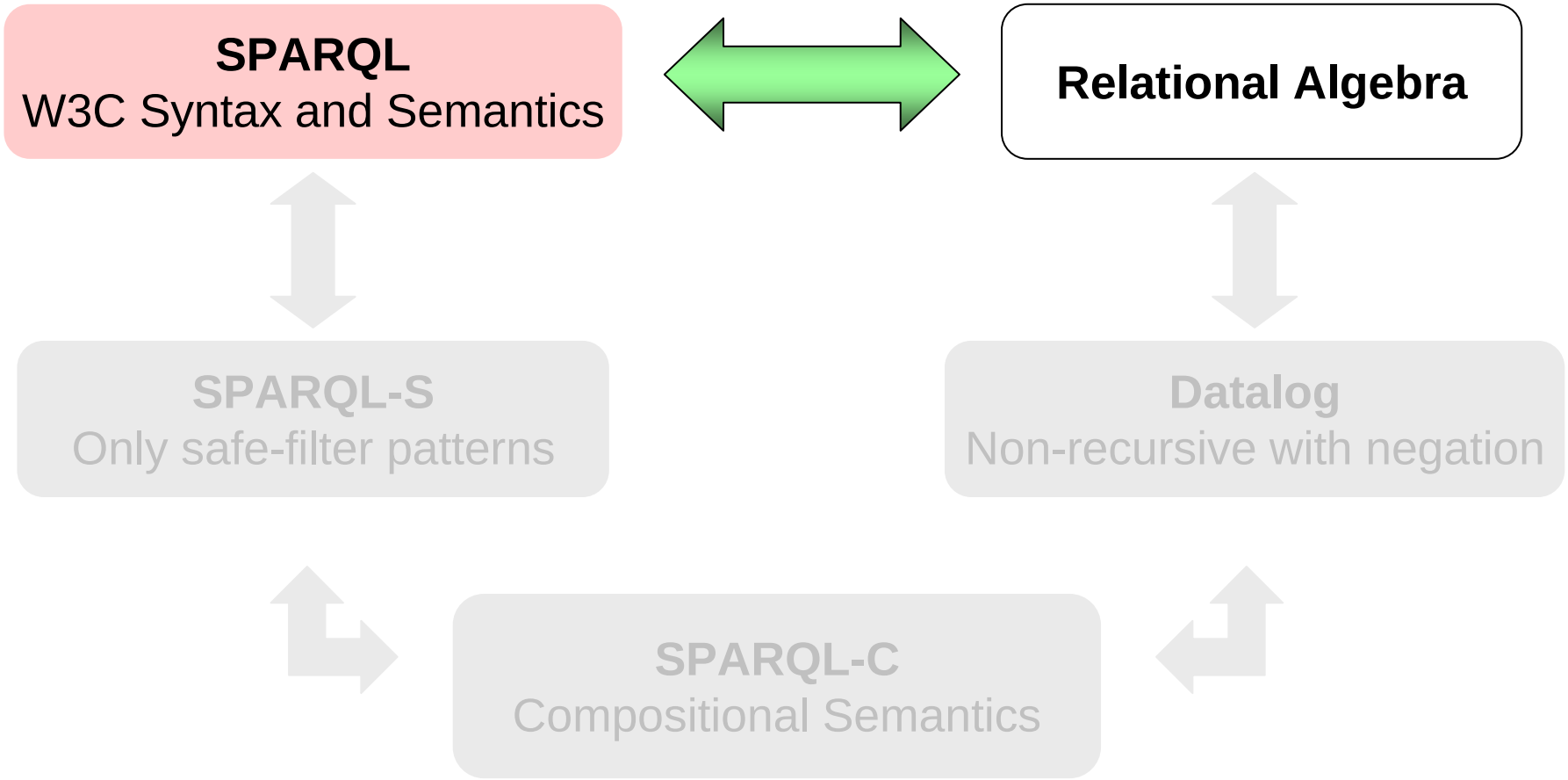
Expressive power of SPARQL : Tour



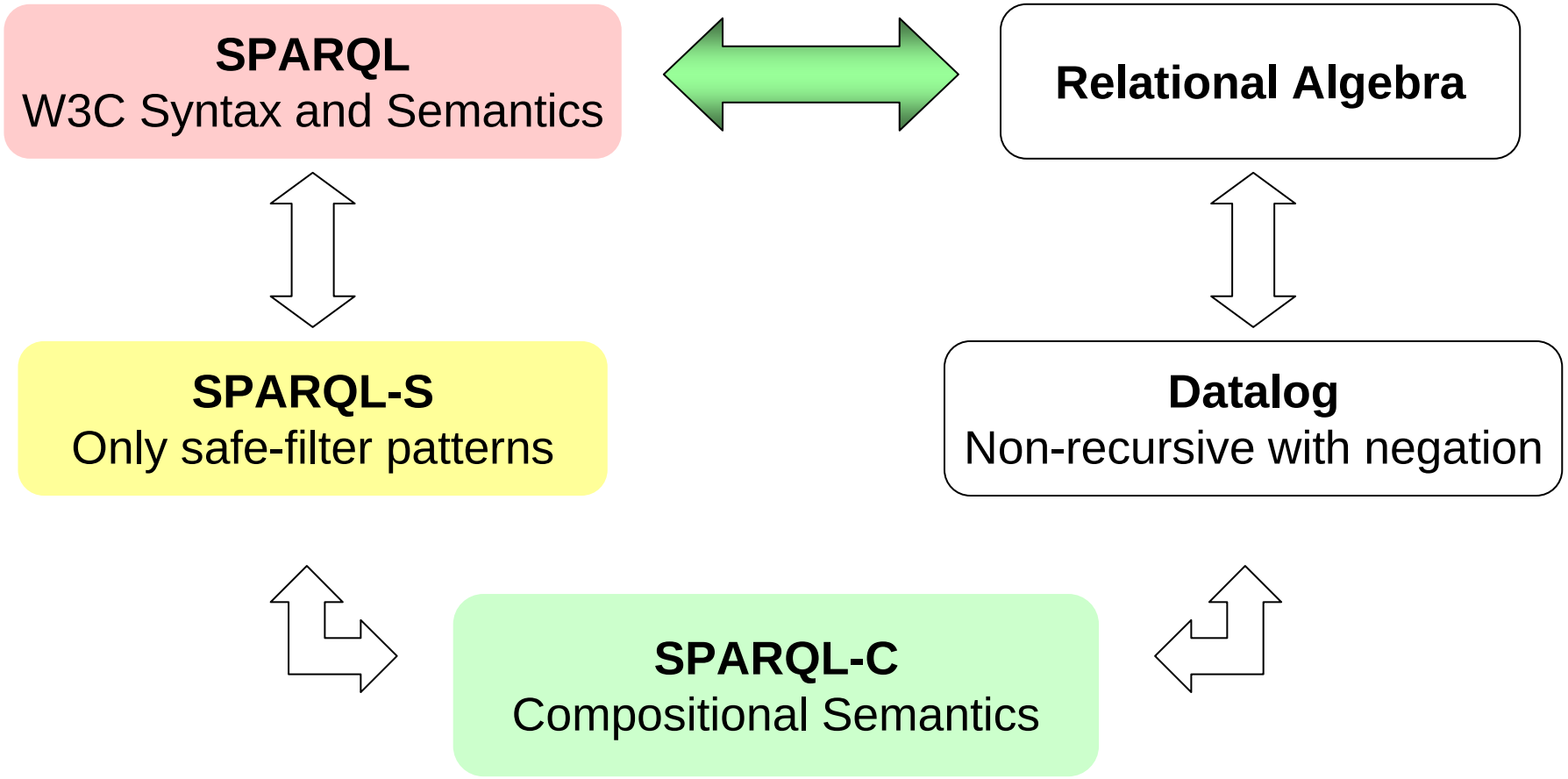
Expressive power of SPARQL : Tour



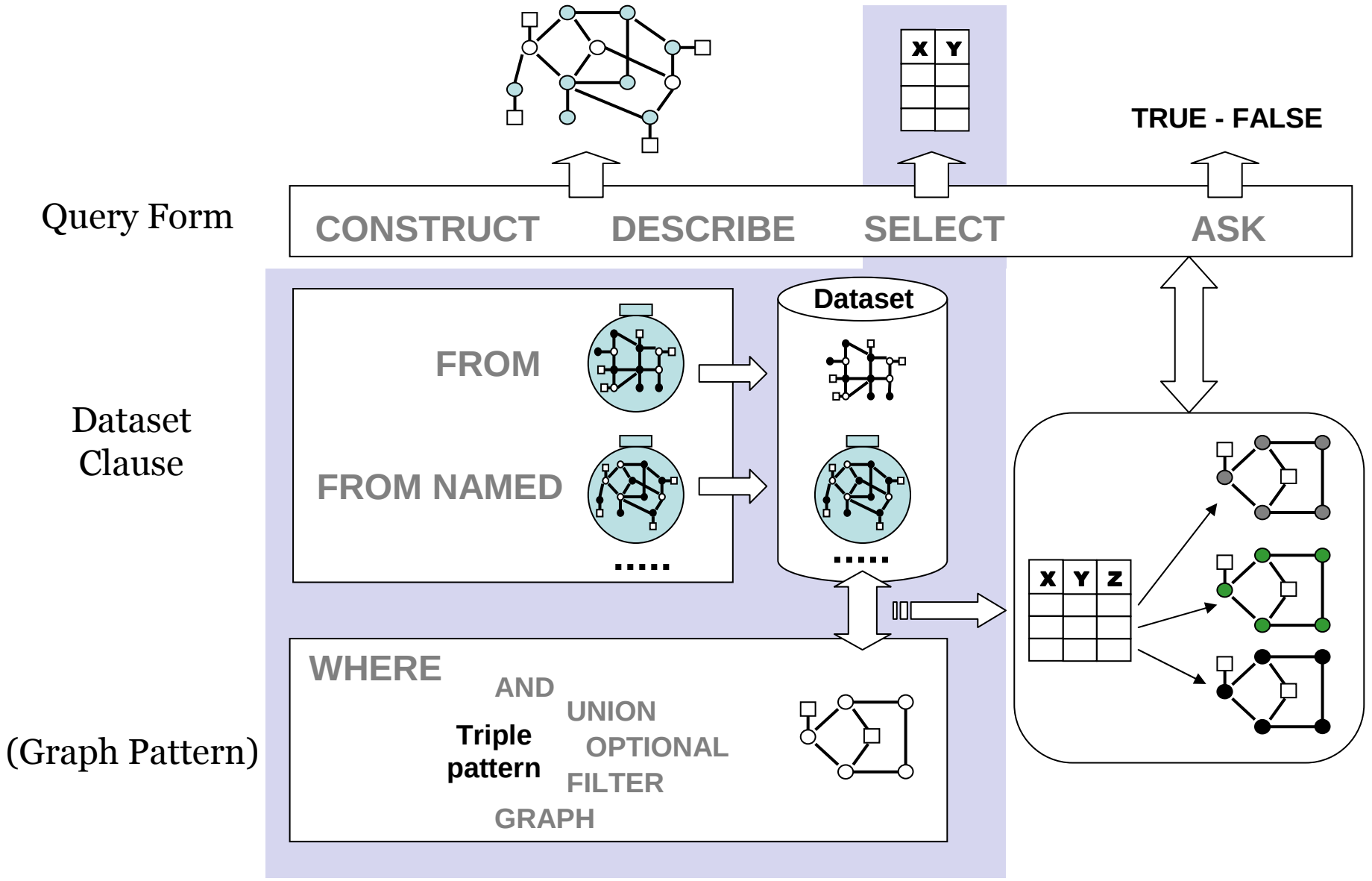
Expressive power of SPARQL : Tour



Expressive power of SPARQL : Tour



SPARQL Query (General structure)



Syntax of SPARQL graph patterns

?X name “George”

Triple pattern (RDF triple + variables)

Syntax of SPARQL graph patterns

?X name “George”	Triple pattern (RDF triple + variables)
{ P_1 . P_2 }	Join of patterns

Syntax of SPARQL graph patterns

?X name “George”	Triple pattern (RDF triple + variables)
{ P_1 . P_2 }	Join of patterns
{ P_1 OPTIONAL { P_2 } }	Optional patterns

Syntax of SPARQL graph patterns

?X name “George”	Triple pattern (RDF triple + variables)
{ P_1 . P_2 }	Join of patterns
{ P_1 OPTIONAL { P_2 } }	Optional patterns
{ P_1 } UNION { P_2 }	Union of patterns

Syntax of SPARQL graph patterns

?X name “George”	Triple pattern (RDF triple + variables)
{ P_1 . P_2 }	Join of patterns
{ P_1 OPTIONAL { P_2 } }	Optional patterns
{ P_1 } UNION { P_2 }	Union of patterns
{ P_1 FILTER C }	Filter conditions over patterns

Syntax of SPARQL graph patterns

Original SPARQL syntax	Algebraic Syntax
?X name “George”	(?X name “George”)
{ P_1 . P_2 }	(P_1 AND P_2)
{ P_1 OPTIONAL { P_2 } }	(P_1 OPT P_2)
{ P_1 } UNION { P_2 }	(P_1 UNION P_2)
{ P_1 FILTER C }	(P_1 FILTER C)

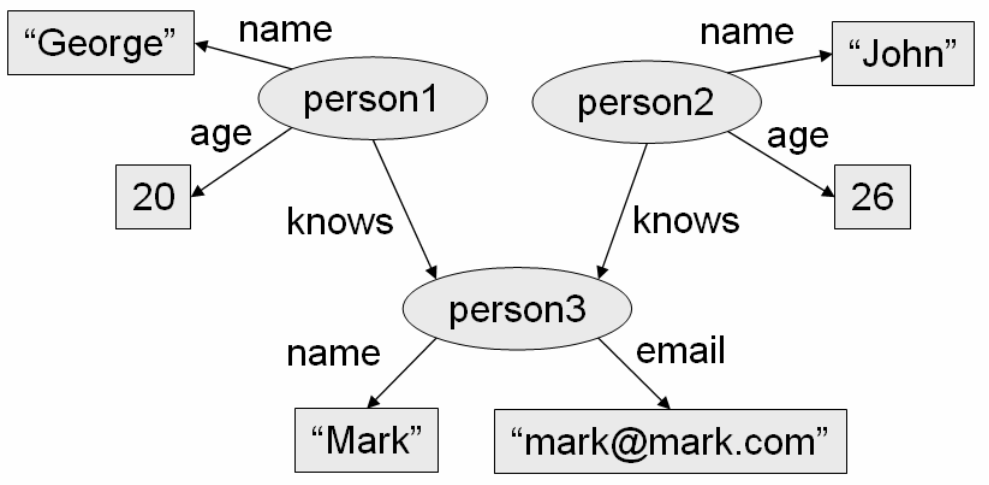
Example of SPARQL query (SELECT)

```
SELECT ?N, ?A, ?E
```

?N	?A	?E

Example of SPARQL query (FROM)

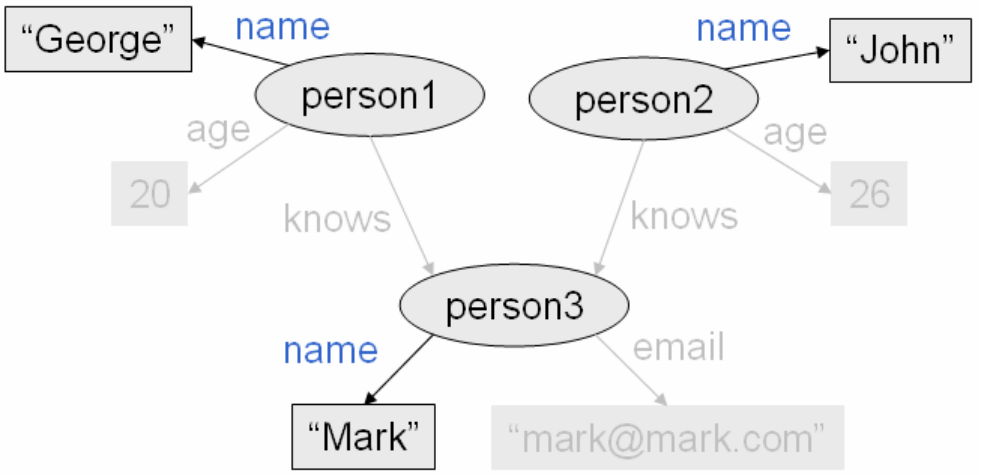
```
SELECT ?N, ?A, ?E
FROM G
```



?N	?A	?E

Example of SPARQL query (Triple pattern)

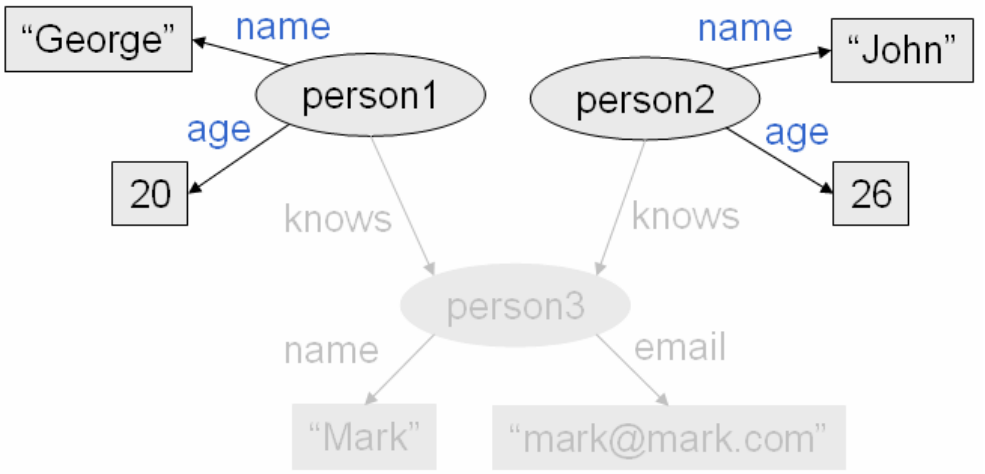
```
SELECT ?N
FROM G
WHERE
    (?X name ?N)
```



?X	?N
person1	"George"
person2	"John"
person3	"Mark"

Example of SPARQL query (AND)

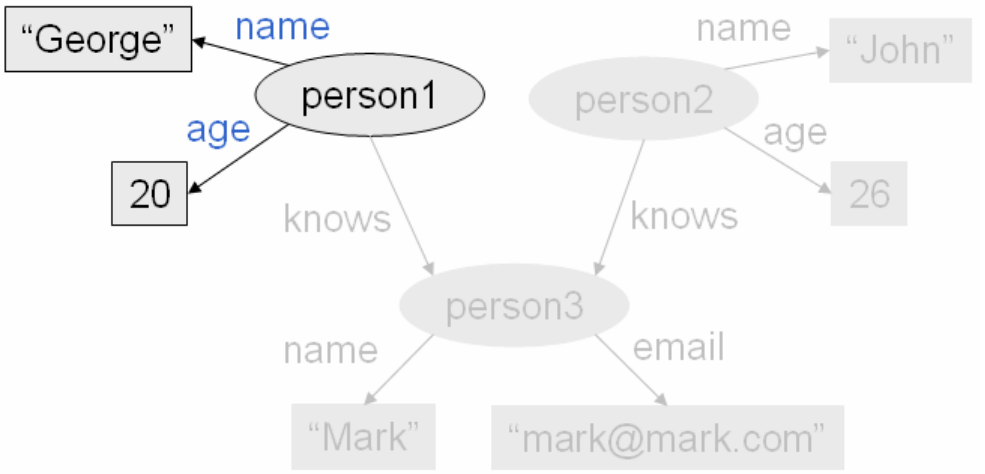
```
SELECT ?N, ?A
FROM G
WHERE
 ( (?X name ?N) AND (?X age ?A) )
```



?X	?N	?A
person1	"George"	20
person2	"John"	26

Example of SPARQL query (FILTER)

```
SELECT ?N, ?A
FROM G
WHERE
 ( (?X name ?N) AND ( (?X age ?A) FILTER (?A < 25) ) )
```

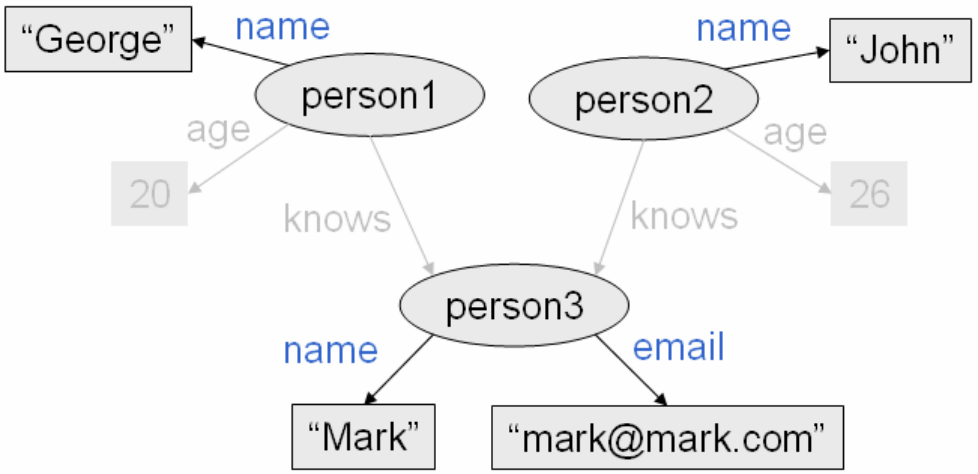


?X	?N	?A
person1	"George"	20

Example of SPARQL query (OPTIONAL)

```
SELECT ?N, ?E
FROM G
WHERE
( (?X,name,?N) AND ( (?X,age,?A) FILTER (?A < 25) ) )
```

((?X name ?N) OPT (?X email ?E))



?X	?N	?A
person1	"George"	20

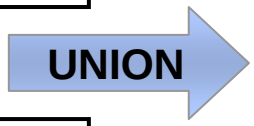
?X	?N	?E
person1	"George"	
person2	"John"	
person3	"Mark"	"Mark@mark.com"

Example of SPARQL query (UNION)

```
SELECT ?N, ?A, ?E
FROM G
WHERE (
 ( (?X name ?N) AND ( (?X age ?A) FILTER (?A < 25) ) )
 UNION
 ( (?X name ?N) OPT (?X email ?E) ) )
```

?X	?N	?A
person1	“George”	20

?X	?N	?E
person1	“George”	
person2	“John”	
person3	“Mark”	“Mark@mark.com”



?N	?A	?E
“George”	20	
“George”		
“John”		
“Mark”		“Mark@mark.com”

W3C Semantics of SPARQL

W3C Recommendation

A.8 Grammar

```
[13]  WhereClause              ::= 'WHERE'? GroupGraphPattern

[20]  GroupGraphPattern        ::= '{' TriplesBlock? ( ( GraphPatternNotTriples | Filter ) '.'? TriplesBlock? )*
                                   '}'
[21]  TriplesBlock             ::= TriplesSameSubject ( '.' TriplesBlock? )?
[22]  GraphPatternNotTriples   ::= OptionalGraphPattern | GroupOrUnionGraphPattern | GraphGraphPattern
[23]  OptionalGraphPattern     ::= 'OPTIONAL' GroupGraphPattern
[24]  GraphGraphPattern        ::= 'GRAPH' VarOrIRIref GroupGraphPattern
[25]  GroupOrUnionGraphPattern ::= GroupGraphPattern ( 'UNION' GroupGraphPattern )*
[26]  Filter                   ::= 'FILTER' Constraint
```

W3C Semantics of SPARQL

W3C Recommendation

A.8 Grammar

```
[13]  WhereClause              ::= 'WHERE'? GroupGraphPattern
[20]  GroupGraphPattern        ::= '{' TriplesBlock? ( ( GraphPatternNotTriples | Filter ) '.'? TriplesBlock? )*
                                   '}'
[21]  TriplesBlock             ::= TriplesSameSubject ( '.' TriplesBlock? )?
[22]  GraphPatternNotTriples   ::= OptionalGraphPattern | GroupOrUnionGraphPattern | GraphGraphPattern
[23]  OptionalGraphPattern     ::= 'OPTIONAL' GroupGraphPattern
[24]  GraphGraphPattern        ::= 'GRAPH' VarOrIRIref GroupGraphPattern
[25]  GroupOrUnionGraphPattern ::= GroupGraphPattern ( 'UNION' GroupGraphPattern )*
[26]  Filter                   ::= 'FILTER' Constraint
```

Transform(syntax form)

If the form is TriplesBlock

```
The result is BGP(list of triple patterns)
```

If the form is GroupOrUnionGraphPattern

```
Let A := undefined

For each element G in the GroupOrUnionGraphPattern
    If A is undefined
        A := Transform(G)
    Else
        A := Union(A, Transform(G))

The result is A
```

W3C Semantics of SPARQL

W3C Recommendation

A.8 Grammar

```
[13]   WhereClause                ::=
[20]   GroupGraphPattern          ::=
[21]   TriplesBlock               ::=
[22]   GraphPatternNotTriples     ::=
[23]   OptionalGraphPattern       ::=
[24]   GraphGraphPattern          ::=
[25]   GroupOrUnionGraphPattern   ::=
[26]   Filter                     ::=
```

TriplesBlock?)*

attern

Transform(syntax form)

If the form is TriplesBlock

```
The result is BGP(list of tri
```

If the form is GroupOrUnionGrap

```
Let A := undefined

For each element G in the Gro
    If A is undefined
        A := Transform(G)
    Else
        A := Union(A, Transfo

The result is A
```

If the form is GroupGraphPattern

We introduce the following symbols:

- Join(Pattern, Pattern)
- LeftJoin(Pattern, Pattern, expression)
- Filter(expression, Pattern)

```
Let FS := the empty set
Let G := the empty pattern, Z, a basic graph pattern

For each element E in the GroupGraphPattern
   If E is of the form FILTER(expr)
       FS := FS set-union {expr}
   If E is of the form OPTIONAL{P}
   Then
       Let A := Transform(P)
       If A is of the form Filter(F, A2)
           G := LeftJoin(G, A2, F)
       else
           G := LeftJoin(G, A, true)
   If E is any other form:
      Let A := Transform(E)
      G := Join(G, A)


If FS is not empty:
  Let X := Conjunction of expressions in FS
  G := Filter(X, G)

The result is G.
```

W3C Semantics of SPARQL

W3C Recommendation

A.8 Grammar

```
[13]   WhereClause              ::=
[20]   GroupGraphPattern        ::=                                  TriplesBlock? )*
[21]   TriplesBlock             ::=
[22]
[23]
[24]
[25]
[26]
```

Transf

```
If
The
If
Let
For
The result is A
```

If the form is GroupGraphPattern

We introduce the following symbols:

- Join(Pattern, Pattern)
- LeftJoin(Pattern, Pattern, expression)

```
  G := Filter(X, G)

The result is G.
```

12.4 SPARQL Algebra

For each symbol in a SPARQL abstract query, we define an operator for evaluation. The SPARQL algebra oper
name are used to evaluate SPARQL abstract query nodes as described in the section "Evaluation Semantics".

Definition: Filter

Let Ω be a multiset of solution mappings and expr be an expression. We define:

Filter(expr, Ω) = { μ | μ in Ω and expr(μ) is an expression that has an effective boolean value of true }

card[Filter(expr, Ω)](μ) = card[Ω](μ)

Definition: Join

Let Ω_1 and Ω_2 be multisets of solution mappings. We define:

Join(Ω_1, Ω_2) = { merge(μ_1, μ_2) | μ_1 in $\Omega_1$and μ_2 in Ω_2, and μ_1 and μ_2 are compatible }

card[Join(Ω_1, Ω_2)](μ) =
  for each merge(μ_1, μ_2), μ_1 in $\Omega_1$and μ_2 in Ω_2 such that μ = merge(μ_1, μ_2),
    sum over (μ_1, μ_2), card[Ω_1](μ_1)*card[Ω_2](μ_2)

W3C Semantics of SPARQL

W3C Recommendation

A.8 Grammar

```
[13]   WhereClause
[20]   GroupGraphPattern
[21]   TriplesBlock
[22]
[23]
[24]
[25]
[26]
```

Transf

```
If
The
If
Let
For
The result is A
```

12.4 SPARQL Algel

For each symbol in a SP
name are used to evalua

Definition: Filter

Let Ω be a multiset of so

Filter(expr, Ω) = { μ | μ in

card[Filter(expr, Ω)](μ) =

Definition: Join

Let Ω_1 and Ω_2 be multisets of solution mappings. We define:

$Join(\Omega_1, \Omega_2) = \{ merge(\mu_1, \mu_2) \mid \mu_1 \text{ in } \Omega_1 \text{and } \mu_2 \text{ in } \Omega_2, \text{ and } \mu_1 \text{ and } \mu_2 \text{ are compatible} \}$

$card[Join(\Omega_1, \Omega_2)](\mu) =$
for each $merge(\mu_1, \mu_2)$, μ_1 in $\Omega_1$and μ_2 in Ω_2 such that $\mu = merge(\mu_1, \mu_2)$,
sum over (μ_1, μ_2), $card[\Omega_1](\mu_1)*card[\Omega_2](\mu_2)$

12.5 Evaluation Semantics

We define eval(D(G), graph pattern) as the evaluation of a graph pattern with respect to a data)*
graph is initially the default graph.

```
D : a dataset
D(G) : D a dataset with active graph G (the one patterns match against)
D[i] : The graph with IRI i in dataset D
D[DFT] : the default graph of D
P, P1, P2 : graph patterns
L : a solution sequence
```

Definition: Evaluation of Filter(F, P)

```
eval(D(G), Filter(F, P)) = Filter(F, eval(D(G),P))
```

Definition: Evaluation of Join(P1, P2)

```
eval(D(G), Join(P1, P2)) = Join(eval(D(G), P1), eval(D(G), P2))
```

Definition: Evaluation of LeftJoin(P1, P2, F)

```
eval(D(G), LeftJoin(P1, P2, F)) = LeftJoin(eval(D(G), P1), eval(D(G), P2), F)
```

```
G := Filter(X, G)

The result is G.
```

Expressive power of SPARQL : Tour

SPARQL
W3C Syntax and Semantics

SPARQL-S
Only safe-filter patterns

Notion of Expressive Power

- A query is a function from the set of input data to the set of output data.
- The expressive power of a query language is given by the set of queries it can express.

Notion of Expressive Power

- A query is a function from the set of input data to the set of output data.
- The expressive power of a query language is given by the set of queries it can express.

Definition (Equivalence of languages)

Two query languages L_1 and L_2 have the same expressive power if they can express the same queries.

Notion of Expressive Power

- A query is a function from the set of input data to the set of output data.
- The expressive power of a query language is given by the set of queries it can express.

Definition (Equivalence of languages)

Two query languages L_1 and L_2 have the same expressive power if they can express the same queries.

(If the languages operate over different data inputs and outputs, have to normalize them before.)

SPARQL-S: Accepting only Safe Patterns

What is the meaning of (P FILTER C) when $\text{var}(C) \not\subseteq \text{var}(P)$ (non-safe filters)?

SPARQL-S: Accepting only Safe Patterns

What is the meaning of (P FILTER C) when $\text{var}(C) \not\subseteq \text{var}(P)$ (non-safe filters)?

Example

Possible meanings of (?X name ?Y) FILTER (?Z $>$ 3)

SPARQL-S: Accepting only Safe Patterns

What is the meaning of (P FILTER C) when $\text{var}(C) \not\subseteq \text{var}(P)$ (non-safe filters)?

Example

Possible meanings of (?X name ?Y) FILTER (?Z $>$ 3)

1. Non-defined variable ?Z. (Error, False, empty set)

SPARQL-S: Accepting only Safe Patterns

What is the meaning of (P FILTER C) when $\text{var}(C) \not\subseteq \text{var}(P)$ (non-safe filters)?

Example

Possible meanings of (?X name ?Y) FILTER (?Z $>$ 3)

1. Non-defined variable ?Z. (Error, False, empty set)
2. All values of ?X, ?Y, ?Z such that the expression matches.

SPARQL-S: Accepting only Safe Patterns

What is the meaning of (P FILTER C) when $\text{var}(C) \not\subseteq \text{var}(P)$ (non-safe filters)?

Example

Possible meanings of (?X name ?Y) FILTER (?Z $>$ 3)

1. Non-defined variable ?Z. (Error, False, empty set)
2. All values of ?X, ?Y, ?Z such that the expression matches.
3. W3C uses the following:
   - IF the expression is inside an optional, e.g. P OPT ((?X name ?Y) FILTER (?Z >3)) and variable ?Z occurs in P, THEN (2.)
   - ELSE (1.)

SPARQL-S: Accepting only Safe Patterns

- Patterns with non-safe filter are rare cases.

SPARQL-S: Accepting only Safe Patterns

- Patterns with non-safe filter are rare cases.
- Patterns with non-safe filters are simulable with safe ones.

SPARQL-S: Accepting only Safe Patterns

- Patterns with non-safe filter are rare cases.
- Patterns with non-safe filters are simulable with safe ones.

Why not avoid them?

SPARQL-S: Accepting only Safe Patterns

- Patterns with non-safe filter are rare cases.
- Patterns with non-safe filters are simulable with safe ones.

Why not avoid them?

Theorem

SPARQL and SPARQL-S have the same expressive power.

SPARQL-S: Accepting only Safe Patterns

- Patterns with non-safe filter are rare cases.
- Patterns with non-safe filters are simulable with safe ones.

Why not avoid them?

Theorem

SPARQL and SPARQL-S have the same expressive power.

Proof idea

- *There is generic procedure to translate non-safe queries into equivalent safe queries.*
- *It uses case-by-case W3C evaluation rules for non-safe queries.*

SPARQL-S: Schema of translation from SPARQL

Proof idea

Given pattern P, define filter-safe pattern $\mathsf{s}(P)$ *recursively:*

SPARQL-S: Schema of translation from SPARQL

Proof idea

Given pattern P, define filter-safe pattern $\mathsf{s}(P)$ *recursively:*

- *Works as the identity for most patterns*

SPARQL-S: Schema of translation from SPARQL

Proof idea

Given pattern P, define filter-safe pattern $\mathrm{s}(P)$ *recursively:*

- *Works as the identity for most patterns*
- *Special Case 1: P is* $(P_1 \,\mathrm{OPT}(P_2 \,\mathrm{FILTER}\, C))$

$$\mathrm{s}(P) \leftarrow (s(P_1) \,\mathrm{OPT}((s(P_1) \,\mathrm{AND}\, s(P_2)) \,\mathrm{FILTER}\, C))$$

SPARQL-S: Schema of translation from SPARQL

Proof idea

Given pattern P, define filter-safe pattern $\mathrm{s}(P)$ *recursively:*

- *Works as the identity for most patterns*
- *Special Case 1: P is* $(P_1 \,\mathrm{OPT}(P_2 \,\mathrm{FILTER}\, C))$

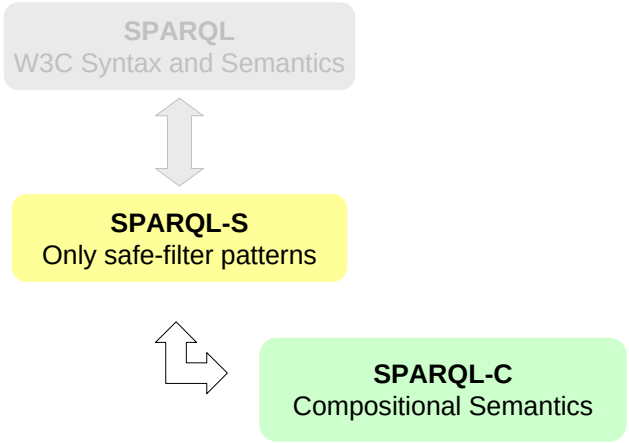
$$\mathrm{s}(P) \leftarrow (s(P_1) \,\mathrm{OPT}((s(P_1) \,\mathrm{AND}\, s(P_2)) \,\mathrm{FILTER}\, C))$$

- *Special Case 2:* $(P_1 \,\mathrm{FILTER}\, C)$ *with* $\mathrm{var}(C) \not\subseteq \mathrm{var}(P_1)$

  For each $X \in \mathrm{var}(C)$ *and not in* $\mathrm{var}(P_1)$ *replace:*

  - *conditions* $(X = a)$ *or* $(X = Y)$ *by error*
    (for ex. $\mathrm{bound}(d)$*, for d constant.)*
  - *condition* $\mathrm{bound}(X)$ *by false.*

Expressive power of SPARQL : Tour



SPARQL-C: SPARQL with compositional semantics

Desiderata for semantics:

SPARQL-C: SPARQL with compositional semantics

Desiderata for semantics:

- Compositional approach: The meaning of an expression is determined by the meaning of its parts and the way they are combined.

SPARQL-C: SPARQL with compositional semantics

Desiderata for semantics:

- Compositional approach: The meaning of an expression is determined by the meaning of its parts and the way they are combined.
- Denotational approach: Meaning of expressions is formalized by assigning mathematical objects which describe the meaning.

SPARQL-C: SPARQL with compositional semantics

Desiderata for semantics:

- Compositional approach: The meaning of an expression is determined by the meaning of its parts and the way they are combined.
- Denotational approach: Meaning of expressions is formalized by assigning mathematical objects which describe the meaning.

SPARQL-C has a denotational and compositional semantics.

SPARQL-C Semantics Overview: Building blocks

Definition

- A mapping is a partial function from variables to RDF terms.
- The evaluation of a triple t is the set of mappings that make t to match the graph

SPARQL-C Semantics Overview: Building blocks

Definition

- A mapping is a partial function from variables to RDF terms.
- The evaluation of a triple t is the set of mappings that make t to match the graph

Bag Semantics

- Multisets (bags) instead of set of mappings
- SPARQL uses bag semantics
- Not well understood from a theoretical point of view

SPARQL-C Semantics Overview: Building blocks

Definition

- A mapping is a partial function from variables to RDF terms.
- The evaluation of a triple t is the set of mappings that make t to match the graph

Bag Semantics

- Multisets (bags) instead of set of mappings
- SPARQL uses bag semantics
- Not well understood from a theoretical point of view

In this talk will avoid bag semantics details.

SPARQL-C Semantics Overview: Operations

Let M_1 and M_2 be sets of mappings:

Definition

Join	: $M_1 \bowtie M_2$
Difference	: $M_1 \setminus M_2$
Union	: $M_1 \cup M_2$
Left Outer Join	: $M_1 \mathbin{⟕} M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2)$

SPARQL-C Semantics Overview: Operations

Let M_1 and M_2 be sets of mappings:

Definition

$$\begin{array}{cc} \text{Join} & : M_1 \bowtie M_2 \\ \text{Difference} & : M_1 \setminus M_2 \\ \text{Union} & : M_1 \cup M_2 \\ \text{Left Outer Join} & : M_1 \mathbin{⟕} M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2) \end{array}$$

Definition

Given P_1, P_2 graph patterns and D an RDF graph:

$$\begin{array}{ccc} [\![P_1 \text{ AND } P_2]\!]_D & \rightarrow & [\![P_1]\!]_D \bowtie [\![P_2]\!]_D \\ [\![P_1 \text{ UNION } P_2]\!]_D & \rightarrow & [\![P_1]\!]_D \cup [\![P_2]\!]_D \\ [\![P_1 \text{ OPT } P_2]\!]_D & \rightarrow & [\![P_1]\!]_D \mathbin{⟕} [\![P_2]\!]_D \end{array}$$

SPARQL-C Semantics Overview: FILTERs

In a pattern (P FILTER C), the filter expression C is a Boolean combination of atoms.

Definition

$$[\![P \text{ FILTER } C]\!] = \{ \mu \in [\![P]\!] : \mu \models C \}$$

$$= \text{Set of mappings in } [\![P]\!] \text{ that satisfy } C.$$

Makes sense only if $\text{var}(C) \subseteq \text{var}(P)$ (safe filters).

SPARQL-S is equivalent to SPARQL-C

Theorem

SPARQL-S and SPARQL-C have the same expressive power.

SPARQL-S is equivalent to SPARQL-C

Theorem

SPARQL-S and SPARQL-C have the same expressive power.

Proof idea

- *Check case by case both semantics coincide (the algorithmic for SPARQL-S and the compositional for SPARQL-C).*
- *The only non-trivial case is the semantics of patterns of the form* $(P_1 \text{ OPT } (P_2 \text{ FILTER } C)$.

SPARQL-S is equivalent to SPARQL-C

Theorem

SPARQL-S and SPARQL-C have the same expressive power.

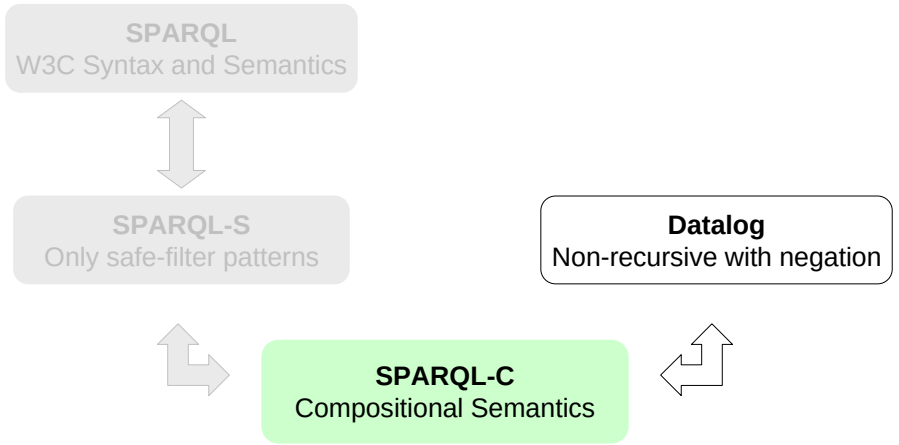
Proof idea

- *Check case by case both semantics coincide (the algorithmic for SPARQL-S and the compositional for SPARQL-C).*
- *The only non-trivial case is the semantics of patterns of the form* $(P_1 \text{ OPT } (P_2 \text{ FILTER } C)$.

Corollary

SPARQL-S has compositional semantics.

Expressive power of SPARQL : Tour



SPARQL-C to Datalog

(Already known in the literature; cf. A. Polleres)

- Represent RDF triples and terms as Datalog facts.
- Represent SPARQL mappings as Datalog substitutions
  In particular, represent the unbounded value in a mapping by the null value.
- Represent each graph pattern as Datalog rules.

SPARQL-C to Datalog

Example (Transformation of AND)

Given the graph pattern

$$(?X \text{ name } ?N) \text{ AND } (?X \text{ age } ?A),$$

it is transformed in the set of rules

1. $P_1(?X, ?N) \leftarrow \mathit{triple}(g, ?X, \mathit{name}, ?N)$
2. $P_2(?X, ?A) \leftarrow \mathit{triple}(g, ?X, \mathit{age}, ?A)$
3. $P(?X, ?N, ?A) \leftarrow$
   $P_1(?X_1, ?N) \wedge P_2(?X_2, ?N) \wedge \mathit{comp}(?X_1, ?X_2, ?X)$

SPARQL-C to Datalog

Example (Transformation of AND)

Given the graph pattern

$$(?X \text{ name } ?N) \text{ AND } (?X \text{ age } ?A),$$

it can be transformed in the set of rules

1. $P_1(?X, ?N) \leftarrow triple(g, ?X, name, ?N)$
2. $P_2(?X, ?A) \leftarrow triple(g, ?X, age, ?A)$
3. $P(?X, ?N, ?A) \leftarrow$
   $P_1(?X_1, ?N) \wedge P_2(?X_2, ?N) \wedge comp(?X_1, ?X_2, ?X)$

SPARQL-C to Datalog

Example (Transformation of AND)

Given the graph pattern

$$(?X \text{ name } ?N) \text{ AND } (?X \text{ age } ?A),$$

it can be transformed in the set of rules

1. $P_1(?X, ?N) \leftarrow \mathit{triple}(g, ?X, \mathit{name}, ?N)$
2. $P_2(?X, ?A) \leftarrow \mathit{triple}(g, ?X, \mathit{age}, ?A)$
3. $P(?X, ?N, ?A) \leftarrow$
   $P_1(?X_1, ?N) \wedge P_2(?X_2, ?N) \wedge \mathit{comp}(?X_1, ?X_2, ?X)$

SPARQL-C to Datalog

Example (Transformation of AND)

Given the graph pattern

$$(?X \text{ name } ?N) \text{ AND } (?X \text{ age } ?A),$$

it can be transformed in the set of rules

1. $P_1(?X, ?N) \leftarrow triple(g, ?X, name, ?N)$
2. $P_2(?X, ?A) \leftarrow triple(g, ?X, age, ?A)$
3. $P(?X, ?N, ?A) \leftarrow$
   $P_1(?X_1, ?N) \wedge P_2(?X_2, ?N) \wedge comp(?X_1, ?X_2, ?X)$

Rules for modeling compatible mappings.

$comp(X, X, X) \leftarrow term(X)$ $\quad$ $comp(X, null, X) \leftarrow term(X)$

$comp(X, X, X) \leftarrow Null(X)$ $\quad$ $comp(null, X, X) \leftarrow term(X)$

Datalog to SPARQL-C

- Represent Datalog facts using RDF triples.

Example

A Datalog fact $p(c_1, \ldots, c_n)$ is described by the set of RDF triples

$$\{(b, \text{predicate}, p), (b, \text{rdf:\_1}, c_1), \ldots, (b, \text{rdf:\_n}, c_n)\}$$

- Direct representation of SPARQL mappings as Datalog subsitutions.

Datalog to SPARQL-C

Given a Datalog rule of the form

$$L \leftarrow L_1 \wedge \cdots \wedge L_s \wedge \neg L_{s+1} \wedge \cdots \wedge \neg L_t \wedge L_1^{eq} \wedge \cdots \wedge L_u^{eq}, \qquad (1)$$

define a function $g(L)$ returning a graph pattern of the form

$$\begin{aligned}(((\cdots((&g(L_1) \text{ AND } \cdots \text{ AND } g(L_s)) \\ &\text{MINUS } g(L_{s+1}))\cdots) \text{ MINUS } g(L_t)) \\ &\text{FILTER}(L_1^{eq} \wedge \cdots \wedge L_u^{eq}))\end{aligned}$$

Datalog to SPARQL-C

Example (Transformation of a Datalog rule)

Given a Datalog rule

$$Q(?N, ?L) \leftarrow name(?X, ?N, ?L) \wedge \neg email(?X, ?E),$$

it can be transformed in the SPARQL query

SELECT ?N,?L
FROM g
WHERE (
((?Y *predicate name*) AND (?Y *rdf:_1* ?X)
AND (?Y *rdf:_2* ?N)
AND (?Y *rdf_3* ?L))
MINUS
((?Z *predicate email*) AND (?Z *rdf:_1* ?X)
AND (?Z *rdf:_2* ?E))
)

Datalog to SPARQL-C

Example (Transformation of a Datalog rule)

Given a Datalog rule

$$Q(?N, ?L) \leftarrow name(?X, ?N, ?L) \wedge \neg email(?X, ?E),$$

it can be transformed in the SPARQL query

```
SELECT ?N,?L
FROM g
WHERE (
( (?Y predicate name) AND (?Y rdf:_1 ?X)
                      AND (?Y rdf:_2 ?N)
                      AND (?Y rdf_3 ?L) )
MINUS
( (?Z predicate email) AND (?Z rdf:_1 ?X)
                       AND (?Z rdf:_2 ?E) )
)
```

Datalog to SPARQL-C

Example (Transformation of a Datalog rule)

Given a Datalog rule

$$Q(?N, ?L) \leftarrow name(?X, ?N, ?L) \wedge \neg email(?X, ?E),$$

it can be transformed in the SPARQL query

```
SELECT ?N,?L
FROM g
WHERE (
( (?Y predicate name) AND (?Y rdf:_1 ?X)
                      AND (?Y rdf:_2 ?N)
                      AND (?Y rdf_3 ?L) )
MINUS
( (?Z predicate email) AND (?Z rdf:_1 ?X)
                       AND (?Z rdf:_2 ?E) )
)
```

Datalog to SPARQL-C

Example (Transformation of a Datalog rule)

Given a Datalog rule

$$Q(?N, ?L) \leftarrow name(?X, ?N, ?L) \wedge \neg email(?X, ?E),$$

it can be transformed in the SPARQL query

$SELECT$?N,?L
$FROM$ g
$WHERE$ (
((?Y $predicate$ $name$) AND (?Y $rdf{:}\_1$?X)
AND (?Y $rdf{:}\_2$?N)
AND (?Y rdf_3 ?L))

$MINUS$
((?Z $predicate$ $email$) AND (?Z $rdf{:}\_1$?X)
AND (?Z $rdf{:}\_2$?E))
)

Datalog to SPARQL-C

Example (Transformation of a Datalog rule)

Given a Datalog rule

$$Q(?N, ?L) \leftarrow name(?X, ?N, ?L) \wedge \neg email(?X, ?E),$$

it can be transformed in the SPARQL query

SELECT ?N,?L
FROM g
WHERE (
((?Y *predicate name*) AND (?Y *rdf:_1* ?X)
AND (?Y *rdf:_2* ?N)
AND (?Y *rdf_3* ?L))
MINUS
((?Z *predicate email*) AND (?Z *rdf:_1* ?X)
AND (?Z *rdf:_2* ?E))
)

Conclusions

Theorem

SPARQL-C and Datalog have the same expressive power.

Conclusions

Theorem

SPARQL-C and Datalog have the same expressive power.

Considering that:

1. SPARQL is equivalent to SPARQL-S;
2. SPARQL-S is equivalent to SPARQL-C;
3. SPARQL-C is equivalent to Datalog; and
4. Datalog is equivalent to Relational Algebra.

Conclusions

Theorem

SPARQL-C and Datalog have the same expressive power.

Considering that:

1. SPARQL is equivalent to SPARQL-S;
2. SPARQL-S is equivalent to SPARQL-C;
3. SPARQL-C is equivalent to Datalog; and
4. Datalog is equivalent to Relational Algebra.

Theorem

SPARQL and Relational Algebra have the same expressive power.

- Results hold for bag and set semantics.

Thank you!

Questions?