Semantics and Complexity of SPARQL

Jorge Pérez[1] Marcelo Arenas[2] Claudio Gutierrez[3]

[1]Universidad de Talca, Chile

[2]Pontificia Universidad Católica de Chile

[3]Universidad de Chile

International Semantic Web Conference 2006

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

In general, in a query we have:

$$H \leftarrow$$

- Head: processing of some variables.

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

In general, in a query we have:

$$H \leftarrow P$$

- Head: processing of some variables.
- Body: pattern matching expression.

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

In general, in a query we have:

$$H \leftarrow P$$

- Head: processing of some variables.
- Body: pattern matching expression.

We focus on P.

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ P1
  P2 }
```

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2 }

  { P3
    P4 }

}
```

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7 }  }

}
```

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
```

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9 }
```

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9
  FILTER ( R ) }
```

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9
  FILTER ( R ) }
```

A formal semantics for SPARQL is needed.

A formal approach would be beneficial

- Clarifying corner cases
- Helping in the implementation process
- Providing sound foundations

A formal semantics for SPARQL is needed.

A formal approach would be beneficial

- Clarifying corner cases
- Helping in the implementation process
- Providing sound foundations ← Our primary interest

A formal semantics for SPARQL is needed.

A formal approach would be beneficial

- Clarifying corner cases
- Helping in the implementation process
- Providing sound foundations ← Our primary interest

In our work:

- A formal compositional semantics (for simple RDF)
- Normal forms for patterns
- Complexity bounds
- Optimization procedures

Outline

Outline

A standard algebraic syntax

- Triple patterns: just triples + variables, without bnodes

original SPARQL syntax	algebraic syntax
`?X :name "john"`	$(?X, \text{name}, \text{john})$

- Graph patterns: full parenthesized algebra

original SPARQL syntax	algebraic syntax
`{ P1 P2 }`	$(P_1 \text{ AND } P_2)$
`{ P1 OPTIONAL { P2 }}`	$(P_1 \text{ OPT } P_2)$
`{ P1 } UNION { P2 }`	$(P_1 \text{ UNION } P_2)$
`{ P1 FILTER ( R ) }`	$(P_1 \text{ FILTER } R)$

A standard algebraic syntax (cont.)

- Explicit precedence/association

Example

```
{ t1
  t2
  OPTIONAL { t3 }
  OPTIONAL { t4 }
  t5
}
```

$$((((\, t_1 \text{ AND } t_2 \,) \text{ OPT } t_3 \,) \text{ OPT } t_4 \,) \text{ AND } t_5 \,)$$

Mappings: building block for the semantics

Definition

A mapping is a partial function from variables to RDF terms.

The evaluation of a pattern results in a set of mappings.

Mappings: building block for the semantics

Definition

A mapping is a partial function from variables to RDF terms.

The evaluation of a pattern results in a set of mappings.

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

Example

graph	triple	evaluation
$(R_1, \text{name}, \text{john})$		
$(R_1, \text{email}, \text{J@ed.ex})$	$(?X, \text{name}, ?Y)$	
$(R_2, \text{name}, \text{paul})$		

	$?X$	$?Y$
μ_1:	R_1	john
μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

Example

graph

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

triple

$(?X, \text{name}, ?Y)$

evaluation

	$?X$	$?Y$
μ_1:	R_1	john
μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

Example

graph

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

triple

$(?X, \text{name}, ?Y)$

evaluation

	$?X$	$?Y$
μ_1:	R_1	john
μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

Example

graph

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

triple

$(?X, \text{name}, ?Y)$

evaluation

	$?X$	$?Y$
μ_1:	R_1	john
μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

Example

graph

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

triple

$(?X, \text{name}, ?Y)$

evaluation

	$?X$	$?Y$
μ_1:	R_1	john
μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

Example

graph

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

triple

$(?X, \text{name}, ?Y)$

evaluation

	$?X$	$?Y$
μ_1:	R_1	john
μ_2:	R_2	paul

Compatible mappings: mappings that can be merged.

Definition

Mappings are compatibles if they agree in their shared variables.
Compatible mappings can be extended.

Example

	$?X$	$?Y$	$?U$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2

Compatible mappings: mappings that can be merged.

Definition

Mappings are compatibles if they agree in their shared variables.
Compatible mappings can be extended.

Example

	$?X$	$?Y$	$?U$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2

Compatible mappings: mappings that can be merged.

Definition

Mappings are compatibles if they agree in their shared variables.
Compatible mappings can be extended.

Example

	$?X$	$?Y$	$?U$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	

Compatible mappings: mappings that can be merged.

Definition

Mappings are compatibles if they agree in their shared variables.
Compatible mappings can be extended.

Example

	$?X$	$?Y$	$?U$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	

Compatible mappings: mappings that can be merged.

Definition

Mappings are compatibles if they agree in their shared variables.
Compatible mappings can be extended.

Example

	$?X$	$?Y$	$?U$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	
$\mu_1 \cup \mu_3$:	R_1	john	P@edu.ex	R_2

Compatible mappings: mappings that can be merged.

Definition

Mappings are compatibles if they agree in their shared variables.
Compatible mappings can be extended.

Example

	$?X$	$?Y$	$?U$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	
$\mu_1 \cup \mu_3$:	R_1	john	P@edu.ex	R_2

- μ_2 and μ_3 are not compatible

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \setminus M_2$

- mappings in M_1 that cannot be extended with mappings in M_2

Union: $M_1 \cup M_2$

- mappings in M_1 plus mappings in M_2 (the usual set union)

Definition

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

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \setminus M_2$

- mappings in M_1 that cannot be extended with mappings in M_2

Union: $M_1 \cup M_2$

- mappings in M_1 plus mappings in M_2 (the usual set union)

Definition

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

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \setminus M_2$

- mappings in M_1 that cannot be extended with mappings in M_2

Union: $M_1 \cup M_2$

- mappings in M_1 plus mappings in M_2 (the usual set union)

Definition

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

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \setminus M_2$

- mappings in M_1 that cannot be extended with mappings in M_2

Union: $M_1 \cup M_2$

- mappings in M_1 plus mappings in M_2 (the usual set union)

Definition

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

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \setminus M_2$

- mappings in M_1 that cannot be extended with mappings in M_2

Union: $M_1 \cup M_2$

- mappings in M_1 plus mappings in M_2 (the usual set union)

Definition

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

Semantics in terms of operations between evaluations

Let M_1 and M_2 be the evaluation of P_1 and P_2.

Definition

The evaluation of:

$$\begin{array}{rcl} (P_1 \text{ AND } P_2) & \rightarrow & M_1 \bowtie M_2 \\ (P_1 \text{ UNION } P_2) & \rightarrow & M_1 \cup M_2 \\ (P_1 \text{ OPT } P_2) & \rightarrow & M_1 \mathbin{⟕} M_2 \end{array}$$

Semantics in terms of operations between evaluations

Let M_1 and M_2 be the evaluation of P_1 and P_2.

Definition

The evaluation of:

$$
\begin{array}{ccc}
(P_1 \text{ AND } P_2) & \rightarrow & M_1 \bowtie M_2 \\
(P_1 \text{ UNION } P_2) & \rightarrow & M_1 \cup M_2 \\
(P_1 \text{ OPT } P_2) & \rightarrow & M_1 \mathbin{⟕} M_2
\end{array}
$$

Semantics in terms of operations between evaluations

Let M_1 and M_2 be the evaluation of P_1 and P_2.

Definition

The evaluation of:

$$\begin{array}{ccc} (P_1 \text{ AND } P_2) & \rightarrow & M_1 \bowtie M_2 \\ (P_1 \text{ UNION } P_2) & \rightarrow & M_1 \cup M_2 \\ (P_1 \text{ OPT } P_2) & \rightarrow & M_1 \mathbin{⟕} M_2 \end{array}$$

Semantics in terms of operations between evaluations

Let M_1 and M_2 be the evaluation of P_1 and P_2.

Definition

The evaluation of:

$$\begin{array}{ccc} (P_1 \text{ AND } P_2) & \rightarrow & M_1 \bowtie M_2 \\ (P_1 \text{ UNION } P_2) & \rightarrow & M_1 \cup M_2 \\ (P_1 \text{ OPT } P_2) & \rightarrow & M_1 \mathbin{⟕} M_2 \end{array}$$

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{email}, ?E))$

- from the Join
- from the Difference
- from the Union

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{email}, ?E))$$

- from the Join
- from the Difference
- from the Union

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$$((?X, \text{ name}, ?Y) \text{ OPT } (?X, \text{ email}, ?E))$$

$?X$	$?Y$
R_1	john
R_2	paul

- from the Join
- from the Difference
- from the Union

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{email}, ?E))$

$?X$	$?Y$
R_1	john
R_2	paul

- from the Join
- from the Difference
- from the Union

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{email}, ?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?E$
R_1	J@ed.ex

- from the Join
- from the Difference
- from the Union

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{email}, ?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?E$
R_1	J@ed.ex

- from the Join
- from the Difference
- from the Union

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{email}, ?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Join
- from the Difference
- from the Union

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{email}, ?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Join
- from the Difference
- from the Union

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{email}, ?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Join
- from the Difference
- from the Union

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{email}, ?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Join
- from the Difference
- from the Union

Boolean filter expressions (value constraints)

In filter expressions we consider

- equality $=$ among variables and RDF terms
- unary predicate bound
- boolean combinations ($\wedge$, $\vee$, $\neg$)

Satisfaction of value constraints

A mapping satisfies

- $?X = c$ if it gives the value c to variable $?X$
- $?X = ?Y$ if it gives the same value to $?X$ and $?Y$
- bound($?X$) if it is defined for $?X$

Definition

The evaluation of (P FILTER R):

- mappings in the evaluation of P that satisfy R.

Satisfaction of value constraints

A mapping satisfies

- $?X = c$ if it gives the value c to variable $?X$
- $?X = ?Y$ if it gives the same value to $?X$ and $?Y$
- bound($?X$) if it is defined for $?X$

Definition

The evaluation of (P FILTER R):

- mappings in the evaluation of P that satisfy R.

Natural algebraic properties: a simple normal from

- AND and UNION are commutative and associative.
- AND, OPT, and FILTER distribute over UNION.

Theorem (UNION Normal Form)

Every graph pattern is equivalent to one of the form

$$P_1 \text{ UNION } P_2 \text{ UNION } \cdots \text{ UNION } P_n$$

with P_i UNION–free.

Natural algebraic properties: a simple normal from

- AND and UNION are commutative and associative.
- AND, OPT, and FILTER distribute over UNION.

Theorem (UNION Normal Form)

Every graph pattern is equivalent to one of the form

$$P_1 \text{ UNION } P_2 \text{ UNION } \cdots \text{ UNION } P_n$$

with P_i UNION*–free.*

Outline

The evaluation decision problem

INPUT:

A mapping, a graph pattern, and an RDF graph.

OUTPUT:

Is the mapping in the evaluation of the pattern against the graph?

Evaluation of simple patterns is polynomial.

Theorem

For patterns using only AND *and* FILTER *operators,*

the evaluation problem is polynomial:

$O(\textit{size of the pattern} \times \textit{size of the graph})$.

Evaluation of simple patterns is polynomial.

Theorem

For patterns using only AND *and* FILTER *operators,*

the evaluation problem is polynomial:

$O(\textit{size of the pattern} \times \textit{size of the graph})$.

Proof.

- Check that the mapping makes every triple to match.
- Then check that the mapping satisfies the FILTERs.

□

Evaluation including UNION is NP-complete.

Theorem

For patterns using only AND, FILTER *and* UNION *operators,*

the evaluation problem is NP-complete.

Evaluation including UNION is NP-complete.

Theorem

For patterns using only AND, FILTER *and* UNION *operators,*

the evaluation problem is NP-complete.

Proof.

- Reduction from propositional SAT.
- The pattern codifies a propositional formula.
- Using $\neg$ bound to codify negation.

□

Evaluation including UNION is NP-complete.

Theorem

For patterns using only AND, FILTER *and* UNION *operators,*

the evaluation problem is NP-complete.

Proof.

- Reduction from propositional SAT.
- The pattern codifies a propositional formula.
- Using $\neg$ bound to codify negation.

□

Evaluation in general is PSPACE-complete.

Theorem

For general patterns that include OPT *operator,*

the evaluation problem is PSPACE-complete.

Evaluation in general is PSPACE-complete.

Theorem

For general patterns that include OPT *operator,*

the evaluation problem is PSPACE-complete.

Proof.

- Reduction from propositional quantified SAT
- The pattern codifies a quantified formula

$$\forall x_1 \exists x_2 \forall x_3 \cdots \varphi.$$

- Using nested OPTs to codify quantifier alternations.
  (This time, we don't need $\neg$ bound.)

Evaluation in general is PSPACE-complete.

Theorem

For general patterns that include OPT *operator,*

the evaluation problem is PSPACE-complete.

Proof.

- Reduction from propositional quantified SAT
- The pattern codifies a quantified formula

$$\forall x_1 \exists x_2 \forall x_3 \cdots \varphi.$$

- Using nested OPTs to codify quantifier alternations.
  (This time, we don't need $\neg$ bound.)

Evaluation in general is PSPACE-complete.

Theorem

For general patterns that include OPT *operator,*

the evaluation problem is PSPACE-complete.

Proof.

- Reduction from propositional quantified SAT
- The pattern codifies a quantified formula

$$\forall x_1 \exists x_2 \forall x_3 \cdots \varphi.$$

- Using nested OPTs to codify quantifier alternations.
  (This time, we don't need $\neg$ bound.)

Data–complexity is polynomial.

Theorem

When considering the pattern fixed,

the evaluation problem is polynomial.

Data–complexity is polynomial.

Theorem

When considering the pattern fixed,

the evaluation problem is polynomial.

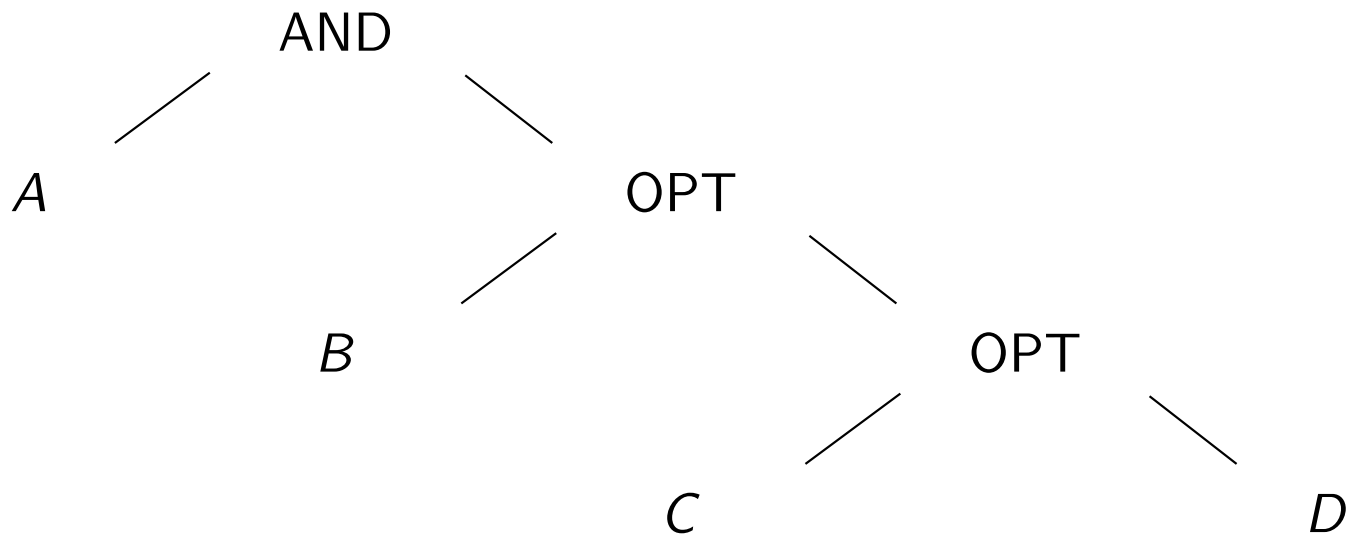
Proof.

From the data–complexity of first–order logic. □

Outline

A procedural semantics: depth–first evaluation

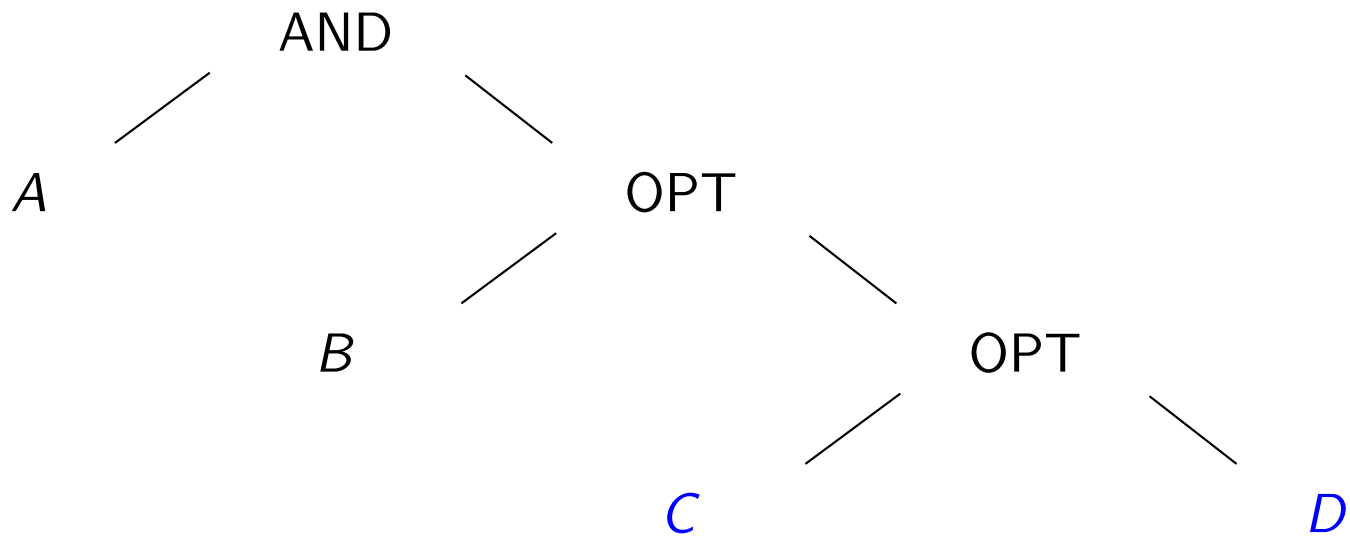
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

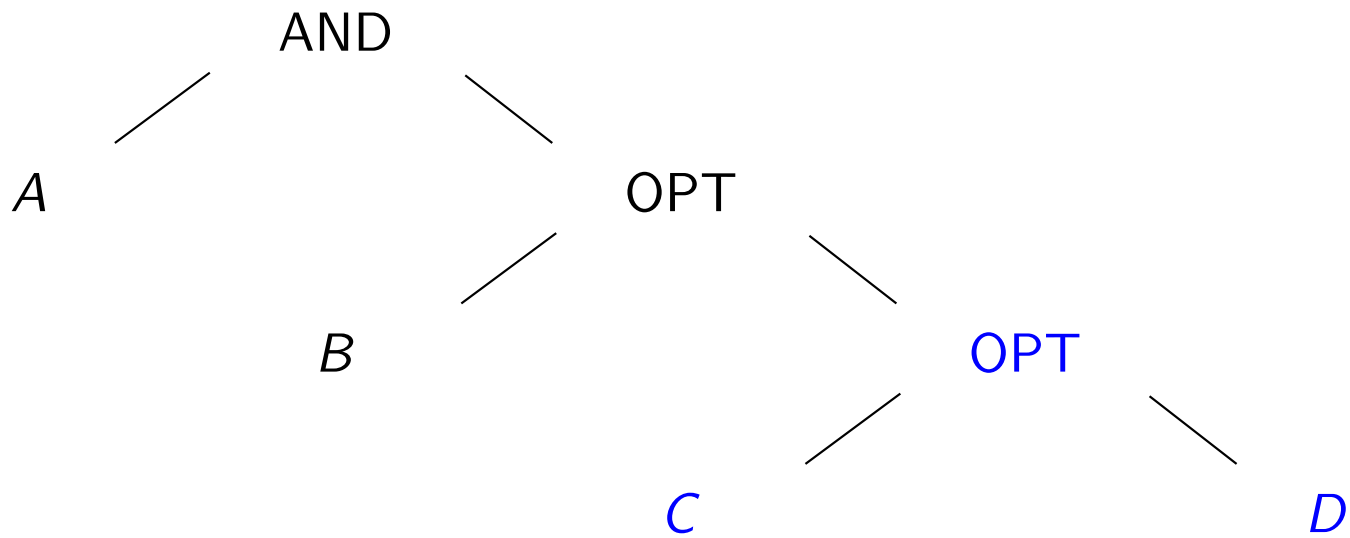
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

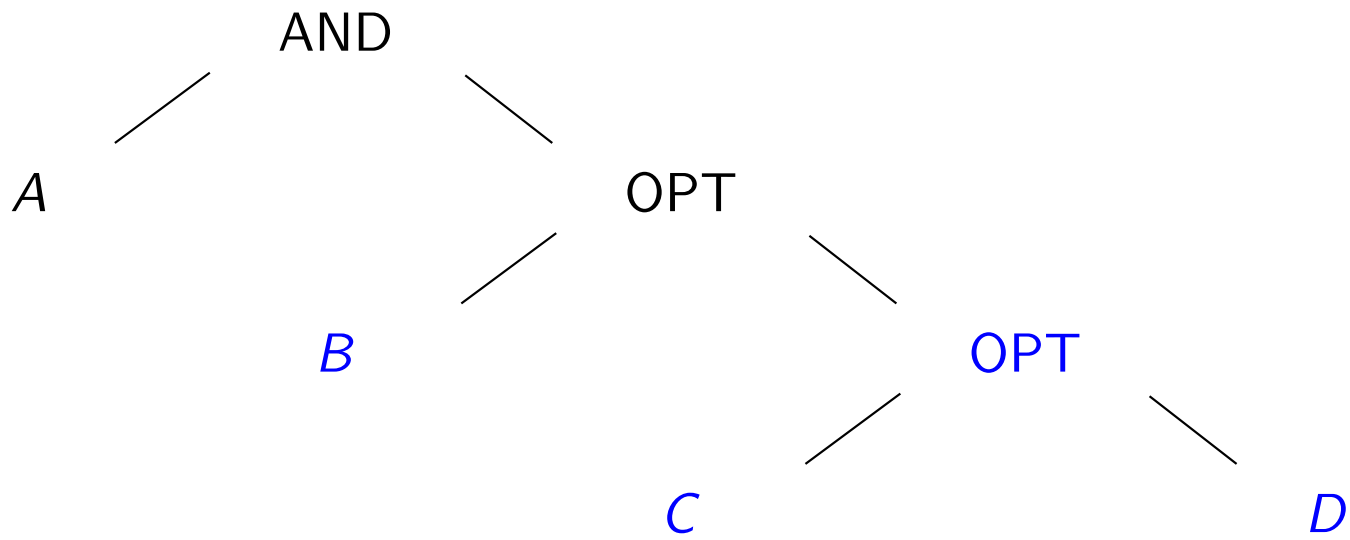
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

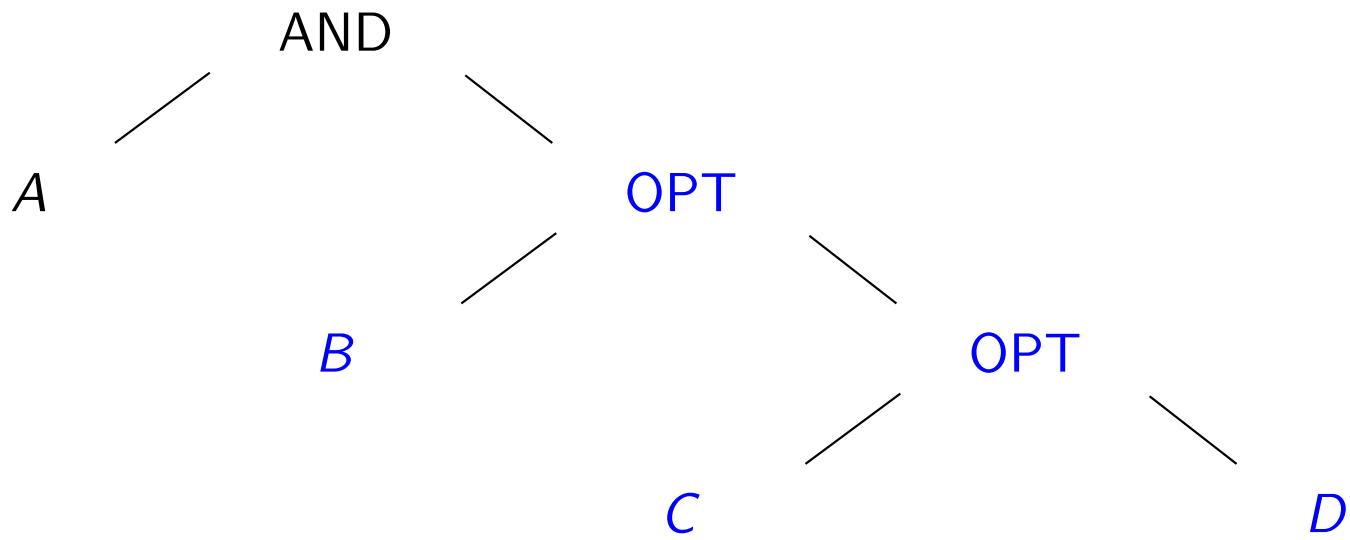
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

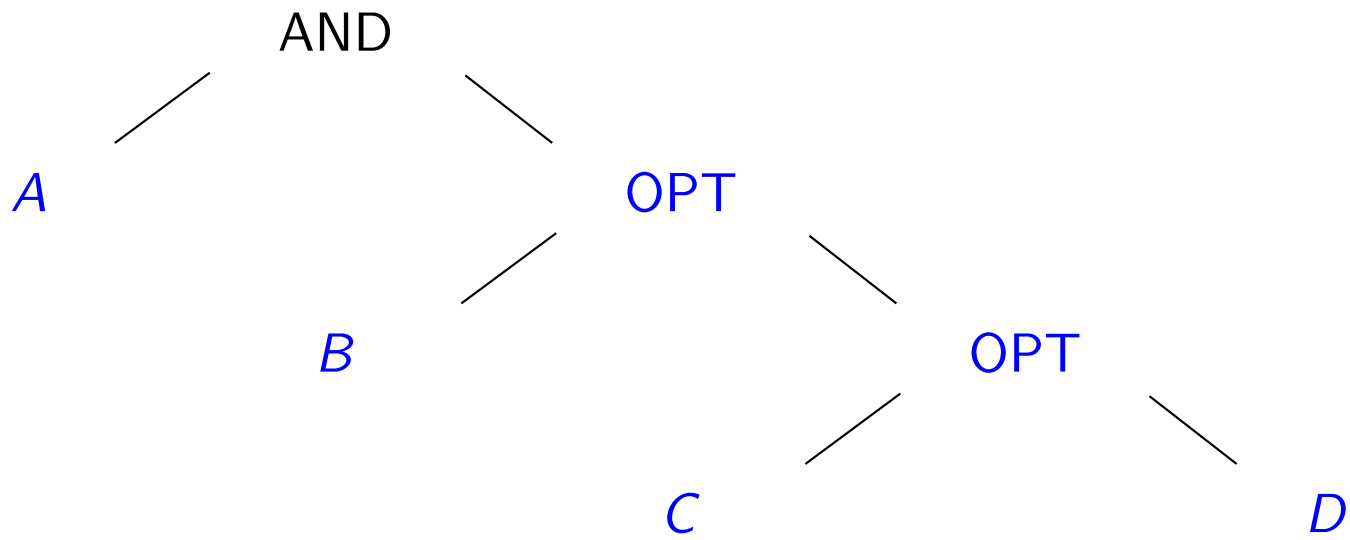
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

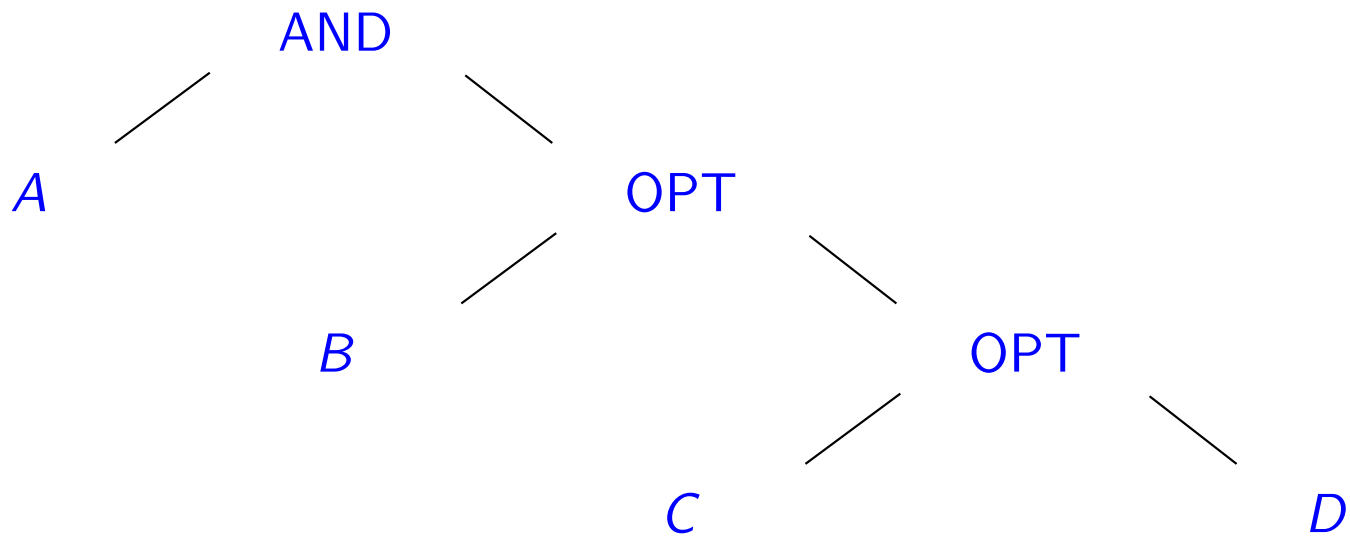
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

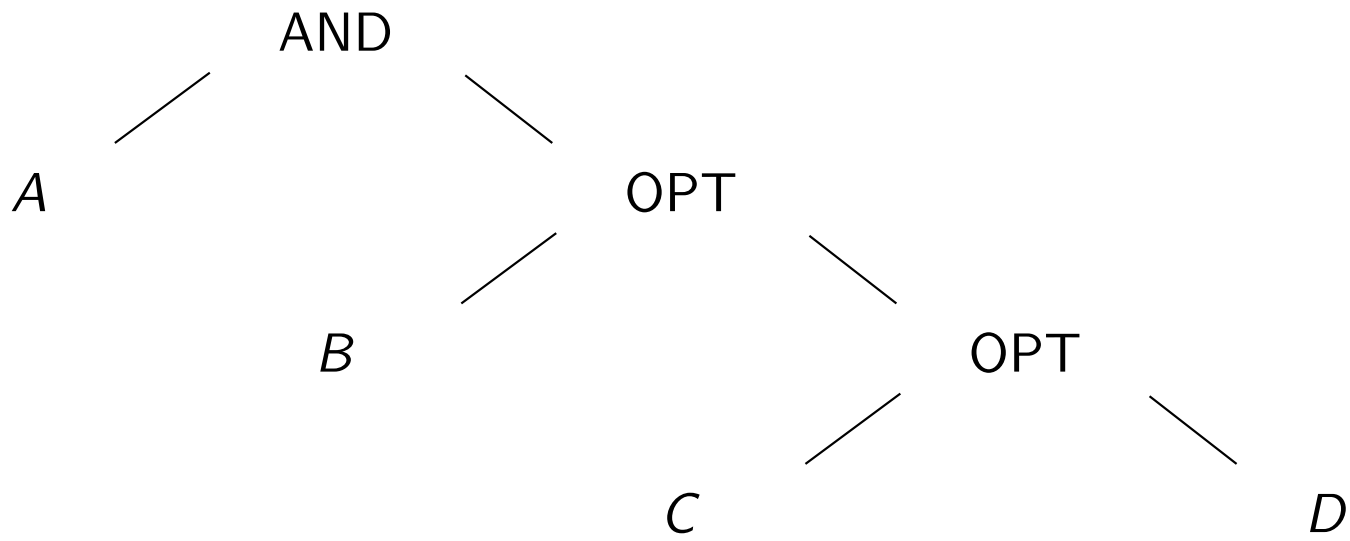
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

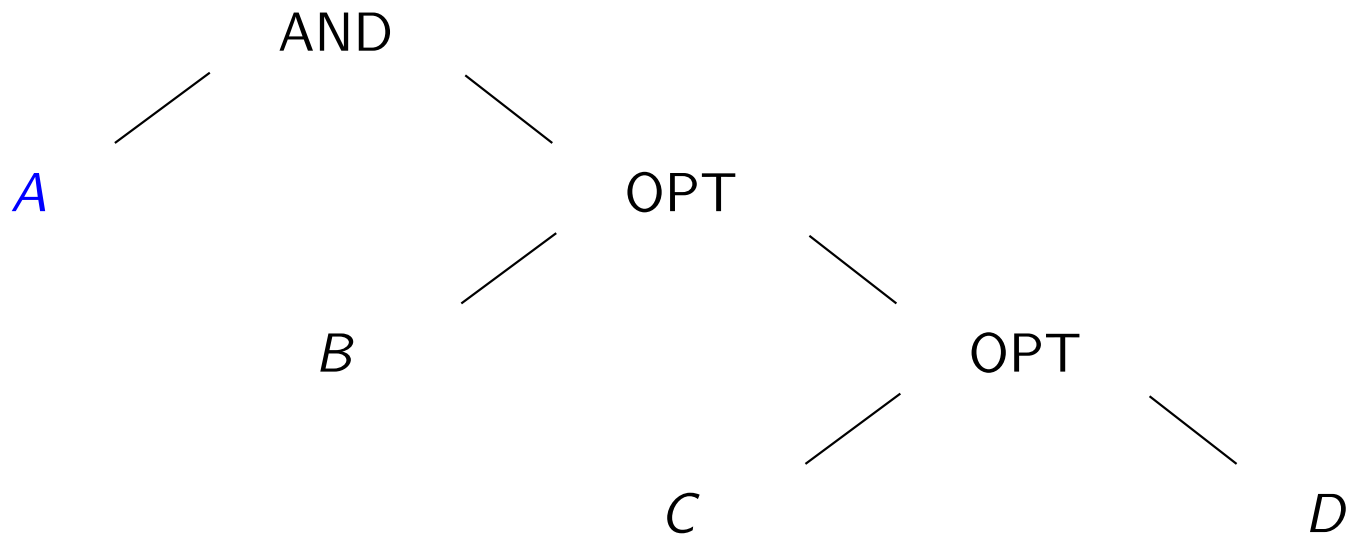
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

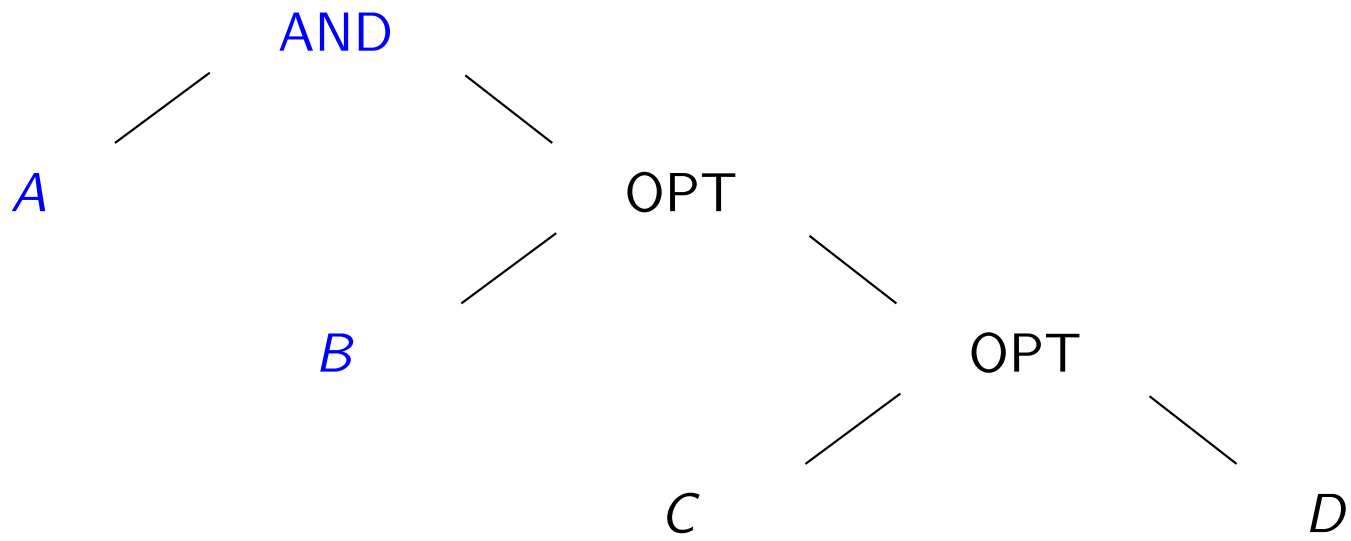
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

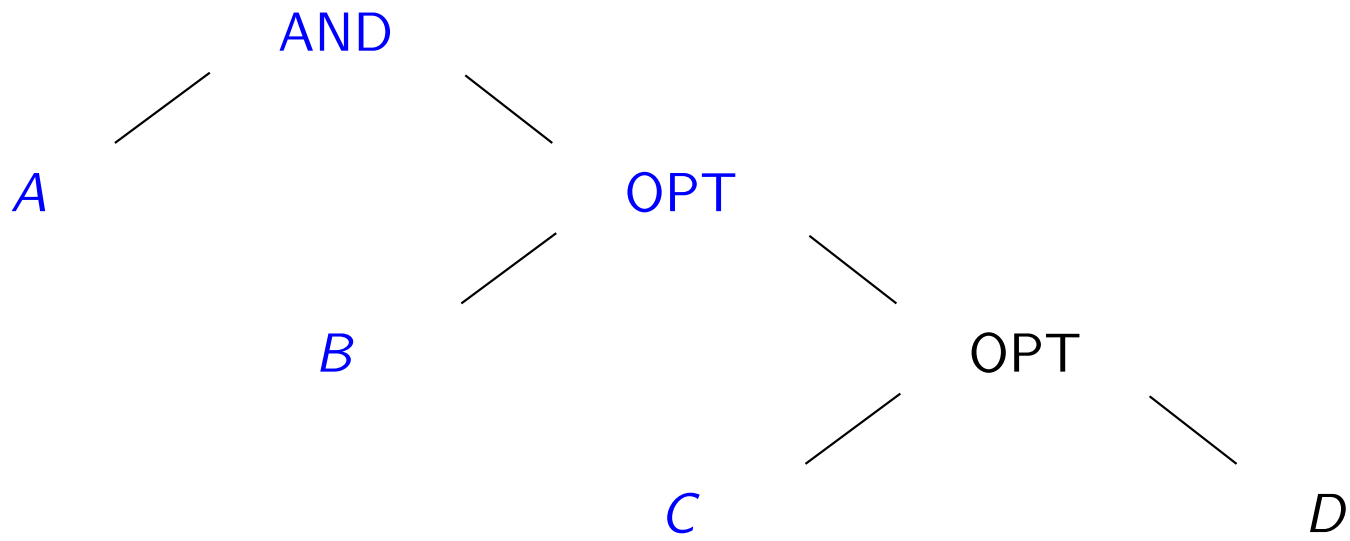
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

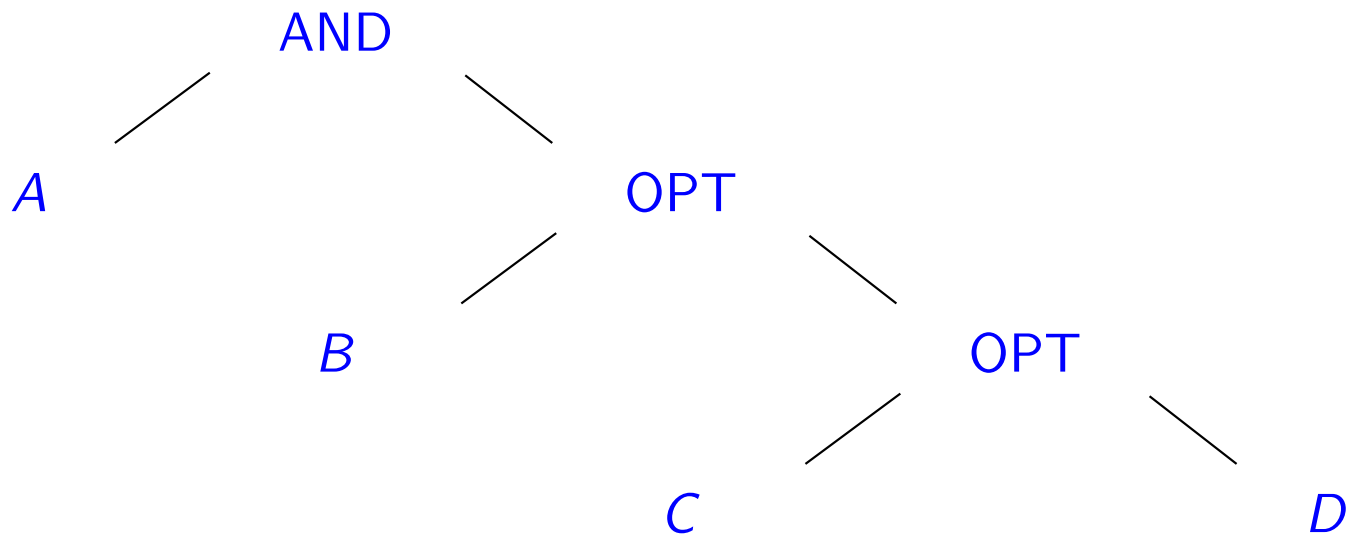
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

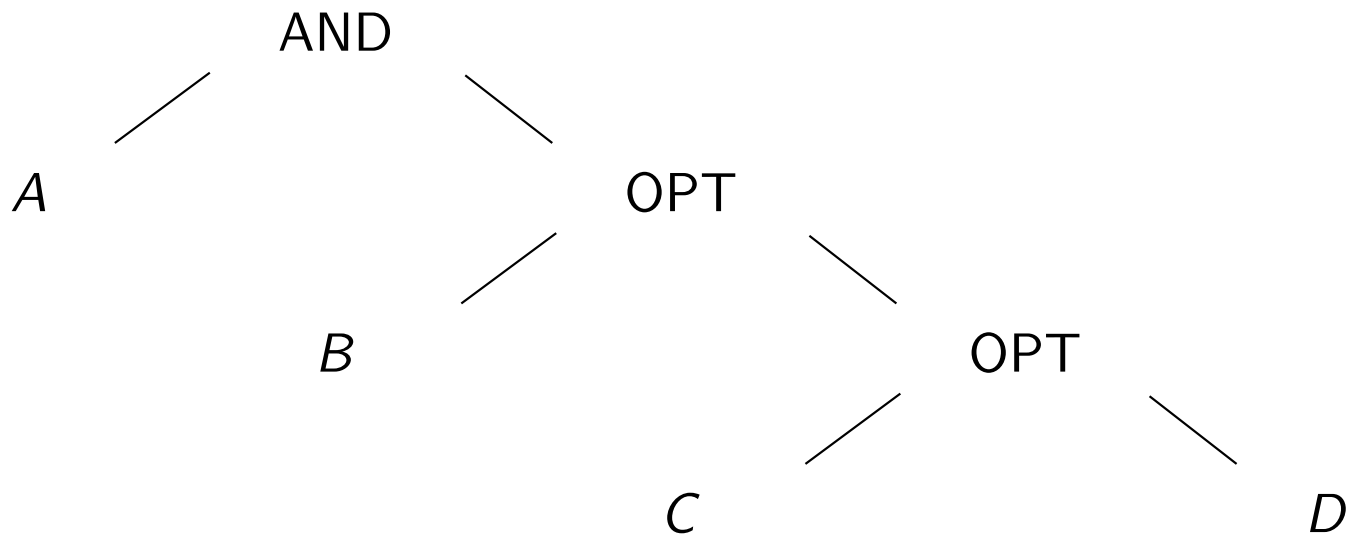
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation

Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- An alternative semantics: a depth–first over the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL
- The two evaluations not always coincide.

A procedural semantics: depth–first evaluation (cont.)

Depth–first evaluation:

- Efficient (greedy): uses intermediate results to avoid some computations.
- non-compositional
- AND of patterns is non-commutative.

A procedural semantics: depth–first evaluation (cont.)

Depth–first evaluation:

- Efficient (greedy): uses intermediate results to avoid some computations.
- non-compositional
- AND of patterns is non-commutative.

A simple condition for both semantics to coincide.

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

$$(\ (\ (?Y, \text{name}, \text{paul})\ \text{OPT}\ (?X, \text{email}, ?E)\ )\quad \text{AND}\quad (?X, \text{name}, \text{john})\ )$$

A simple condition for both semantics to coincide.

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad \underset{\uparrow}{B} \quad) \quad \cdots\cdots\cdots\cdots)$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

$$(\; (\; (?Y, \text{name}, \text{paul}) \; \text{OPT} \; (?X, \text{email}, ?E) \;) \quad \text{AND} \quad (?X, \text{name}, \text{john}) \;)$$

A simple condition for both semantics to coincide.

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$
$$\uparrow \qquad\qquad\qquad\qquad \uparrow \qquad\qquad\qquad \uparrow$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

$(\ (\ (?Y, \text{name}, \text{paul})\ \text{OPT}\ (?X, \text{email}, ?E)\ )\quad \text{AND}\quad (?X, \text{name}, \text{john})\ )$

A simple condition for both semantics to coincide.

Definition

A graph pattern is well–designed iff for every OPT in the pattern

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

(((?Y, name, paul) OPT (?X, email, ?E)) AND (?X, name, john))

A simple condition for both semantics to coincide.

Definition

A graph pattern is well–designed iff for every OPT in the pattern

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

(((?Y, name, paul) OPT (?X, email, ?E)) AND (?X, name, john))

A simple condition for both semantics to coincide.

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

$$(\; (\; (?Y, \text{name}, \text{paul}) \; \text{OPT} \; (?X, \text{email}, ?E) \;) \quad \text{AND} \quad (?X, \text{name}, \text{john}) \;)$$

A simple condition for both semantics to coincide.

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

$$(\; (\; (?Y, \text{name}, \text{paul}) \; \text{OPT} \; (?X, \text{email}, ?E) \;) \quad \text{AND} \quad (?X, \text{name}, \text{john}) \;)$$

A simple condition for both semantics to coincide.

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

$$(\; (\; (?Y, \text{name}, \text{paul}) \; \text{OPT} \; (?X, \text{email}, ?E) \;) \quad \text{AND} \quad (?X, \text{name}, \text{john}) \;)$$

A simple condition for both semantics to coincide (cont.).

Theorem

For well–designed graph patterns

$$\textit{depth–first evaluation} \quad \equiv \quad \textit{compositional semantics}$$

Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
  - null–rejection: the join/outer–join condition must fail in the presence of null.
- SPARQL operations are never null–rejecting
  - by definition of compatible mappings.

Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
  - null–rejection: the join/outer–join condition must fail in the presence of null.
- SPARQL operations are never null–rejecting
  - by definition of compatible mappings.

Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
  - null–rejection: the join/outer–join condition must fail in the presence of null.
- SPARQL operations are never null–rejecting
  - by definition of compatible mappings.

Well–designed graph patterns and optimization

Well–designed patterns suitable for reordering–optimization:

Theorem (OPT Normal Form)

Every well–designed pattern is equivalent to one of the form

$$(\cdots(t_1 \text{ AND } \cdots \text{ AND } t_k) \text{ OPT } O_1)\cdots) \text{ OPT } O_n)$$

with t_i triple pattern, and O_i a pattern of the same form.

Well–designed graph patterns and optimization

Well–designed patterns suitable for reordering–optimization:

Theorem (OPT Normal Form)

Every well–designed pattern is equivalent to one of the form

$$(\cdots(t_1 \text{ AND } \cdots \text{ AND } t_k) \text{ OPT } O_1)\cdots) \text{ OPT } O_n)$$

with t_i triple pattern, and O_i a pattern of the same form.

Summary

- A formal compositional semantics for SPARQL
- Complexity bounds
- Comparisons with a procedural semantics (df evaluation)
- Normalization and initial optimizations procedures

Ongoing work

- Extend to the whole language
- Include entailment for base cases definitions (need a W3C decision first).
- Study other DB aspects of the language

Summary

- A formal compositional semantics for SPARQL
- Complexity bounds
- Comparisons with a procedural semantics (df evaluation)
- Normalization and initial optimizations procedures

Ongoing work

- Extend to the whole language
- Include entailment for base cases definitions (need a W3C decision first).
- Study other DB aspects of the language

Summary

- A formal compositional semantics for SPARQL
- Complexity bounds
- Comparisons with a procedural semantics (df evaluation)
- Normalization and initial optimizations procedures

Ongoing work

- Extend to the whole language
- Include entailment for base cases definitions (need a W3C decision first).
- Study other DB aspects of the language

Summary

- A formal compositional semantics for SPARQL
- Complexity bounds
- Comparisons with a procedural semantics (df evaluation)
- Normalization and initial optimizations procedures

Ongoing work

- Extend to the whole language
- Include entailment for base cases definitions (need a W3C decision first).
- Study other DB aspects of the language

Summary

- A formal compositional semantics for SPARQL
- Complexity bounds
- Comparisons with a procedural semantics (df evaluation)
- Normalization and initial optimizations procedures

Ongoing work

- Extend to the whole language
- Include entailment for base cases definitions (need a W3C decision first).
- Study other DB aspects of the language