Outline

- The Expressive Power of SPARQL
- From SPARQL to Rules (and back)
- Importance (Why we care)
- Future Work
The Expressive Power of SPARQL

Renzo Angles, Claudio Gutierrez

Universidad de Chile
Outline

- **SPARQL**
  - W3C Syntax and Semantics

- **SPARQL-S**
  - Only safe-filter patterns

- **SPARQL-C**
  - Compositional Semantics

- **Relational Algebra**

- **Datalog**
  - Non-recursive with negation
Overview of SPARQLWG

• A SPARQL query is composed of:
  – Query form: SELECT, ASK, CONSTRUCT, DESCRIBE
  – Dataset: FROM, FROM NAMED
  – Graph patterns:
    • Triple patterns: ?X name “George”
    • Join of patterns: \{ P_1 \ . \ P_2 \}
    • Optional patterns: \{ P_1 \ OPTIONAL \ \{ P_2 \} \}
    • Union of patterns: \{ P_1 \} UNION \{ P_2 \}
    • Filter conditions over patterns: \{ P_1 \ FILTER C \}
    • Patterns on named graphs: \{ u GRAPH P_1 \} / \{ ?X GRAPH P_1 \}
Overview of SPARQLWG

Data:
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
_:a foaf:givenName "Alice".
_:b foaf:givenName "Bob".
_:b dc:date "2005-04-04T04:04:04Z"^^xsd:dateTime

Query:
PREFIX foaf: <http://xmlns.com/foaf/0.1/> 
PREFIX dc: <http://purl.org/dc/elements/1.1/> 
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> 
SELECT ?name 
    WHERE { ?x foaf:givenName ?name .
              OPTIONAL { ?x dc:date ?date } .
              FILTER ( !bound(?date) ) }

Above query finds out all people that has foaf:givenName but not dc:date.
Overview of SPARQLWG

- **Issue 1: difference of patterns** \((P_1 \text{ MINUS } P_2)\)
  - Current approach: \(((P_1 \text{ OPT } P_2) \text{ FILTER } (!\text{BOUND(?X)}))\) where \?X\(\in\text{var}(P_2) \backslash \text{var}(P_1)\)
  - Problems:
    - What if \(P_2=(P_3 \text{ OPT } P_4)\)? Only var(\(P_3\)) are relevant.
    - What if \text{var}(P_2) \backslash \text{var}(P_1) = \emptyset\).
  - Solutions:
    - Non-optional variables nov(\(P\)): the variables must be bounded in any mapping of \(P\).
    - Copy pattern \(\Phi(P)\): isomorphic copy of \(P\) whose variables have been renamed according to \(\Phi\) s.t. var(\(P\)) and var(\(\Phi(P)\)) are disjoint.
    - \((P_1 \text{ MINUS } P_2) = ((P_1 \text{ OPT } ((P_2 \text{ AND } \Phi(P_2)) \text{ FILTER } C_1)) \text{ FILTER } C_2)\) where:
      - \(C_1\) is \((?X_1 = ?X_1’ \wedge \ldots \wedge ?X_n = ?X_n’)\) where \(?X_i\in\text{var}(P_2)\) and \(?X_i’= \Phi(?X_i)\) for \(1 \leq i \leq n\).
      - \(C_2\) is \(!\text{bound(?X’)}\) for some \(?X’\in\text{nov(\(\Phi(P_2)\))}\).
  - Example: \(P = ((?X, \text{ name, } ?N) \text{ MINUS } ((?X, \text{ know, } ?Y) \text{ OPT } (?Y, \text{ mail, } ?Z))))\).
    - \(P’ = ((P_1 \text{ OPT } P_2) \text{ FILTER } (!\text{BOUND(?Z)}))\) where \(P_1= (\?X, \text{ name, } \?N)\), \(P_2 = ((\?X, \text{ know, } \?Y) \text{ OPT } (?Y, \text{ mail, } \?Z))\)
    - \(P” = ((P_1 \text{ OPT } P_2) \text{ FILTER } (!\text{BOUND(?Y)}))\)
Overview of SPARQL\textsubscript{WG}

- Issue 2: unsafe filters (P \text{ FILTER} C) where variables in C not present in P.
  - Current approach: context-dependent
    - Scope of C is the global graph pattern
    - Exception: scope of C is local to the OPT pattern when \((P_1 \text{ OPT} (P_2 \text{ FILTER} C))\)
  - Problems: complexity
  - Solution: \(\llangle P \rrangle = \llangle \text{(safe)} P \rrangle\) where \(\text{safe}(P) \in \text{SPARQL}_{\text{WG}}^{\text{Safe}}\) which is the subset of queries having only filter-safe patterns.
SPARQL $\iff$ SPARQL$_S$

- SPARQL$_{WG}$ and SPARQL$_{WG}^{Safe}$ have the same expressive power.
\[ \text{SPARQL}_S \Leftrightarrow \text{SPARQL}_C \]

- **SPARQL\(_C\)**: alternative compositional semantics.
- **SPARQL\(_{WG}^{\text{Safe}}\)** is equivalent to **SPARQL\(_C\)** under bag semantics.
  - Check case by case both semantics coincide

**Compositional semantics**: the meaning of an expression is determined by the meaning of its parts and the way they are combined.

**Bag semantics**: multisets (bags) instead of set of mappings.

Tuesday, February 24, 2009
SPARQLC $\leftrightarrow$ nr-Datalog

- Non-recursive and safe Datalog program: it doesn’t contain any predicate that is recursive in the program and it can only generate a finite number of answers.
- A datalog query $Q = (\prod, L)$ where
  - $\prod$ is a datalog program
  - $L$ is a positive literal
  - $\text{Ans}_d(Q, D)$: answer to $Q$ over database $D = \text{facts}(\prod)$ is the set of substitutions $\{ \theta | \theta(L) \in \text{facts}^*(\prod) \}$ where $\theta(L)$ makes a rule $r$ in $\prod$ true and $L$ is the head of $r$.
- A query language is a quadruple $(\mathcal{E}, \mathcal{D}, \mathcal{S}, \text{eval})$
  - $\mathcal{E}$ is a set of queries
  - $\mathcal{D}$ is a set of databases
  - $\mathcal{S}$ is a set of solutions
  - $\text{eval}: \mathcal{E} \times \mathcal{D} \rightarrow \mathcal{S}$ is the evaluation function
  - Two queries $Q_1, Q_2 \in \mathcal{E}$ are equivalent, $Q_1 = Q_2$, if $\text{eval}(Q_1, D) = \text{eval}(Q_2, D)$ for every $D \in \mathcal{D}$.
- A query language $L_1 = (\mathcal{E}_1, \mathcal{D}_1, \mathcal{S}_1, \text{eval}_1)$ is contained in $L_2 = (\mathcal{E}_2, \mathcal{D}_2, \mathcal{S}_2, \text{eval}_2)$ if
  - There are transformations $\mathcal{T}_D: \mathcal{D}_1 \rightarrow \mathcal{D}_2$, $\mathcal{T}_S: \mathcal{S}_1 \rightarrow \mathcal{S}_2$, $\mathcal{T}_Q: \mathcal{E}_1 \rightarrow \mathcal{E}_2$ such that
  - For all $Q \in \mathcal{E}_1$ and $D \in \mathcal{D}_1$, it satisfies that $\mathcal{T}_S(\text{eval}_1(Q, D)) = \text{eval}_2(\mathcal{T}_Q(Q), \mathcal{T}_D(D))$.
- $L_1$ and $L_2$ are equivalent iff $L_1$ is contained in $L_2$ and $L_2$ is contained in $L_1$. 

Tuesday, February 24, 2009
SPARQL\textsubscript{C} $\iff$ nr-Datalog$\neg$

- **Query languages:**
  - **SPARQL:** $L_s = (\mathcal{Z}_s, \mathcal{D}_s, \mathcal{S}_s, \text{eval}_s)$
  - **Datalog:** $L_d = (\mathcal{Z}_d, \mathcal{D}_d, \mathcal{S}_d, \text{eval}_d)$

- **SPARQL\textsubscript{C} $\Rightarrow$ nr-Datalog$\neg$**
  - RDF datasets as Datalog facts.
  - SPARQL mappings as Datalog substitutions (unbounded value in a mapping by the null value).
  - Graph pattern as Datalog rules.
  - Example: $(?X \text{ name } ?N) \text{ AND } (?X \text{ age } ?A) \Rightarrow$
    - $p_1(?X, ?N) \leftarrow \text{ triple}(g, ?X, \text{name}, ?N)$
    - $p_2(?X, ?A) \leftarrow \text{ triple}(g, ?X, \text{age}, ?A)$
    - $p(?X, ?N, ?A) \leftarrow p_1(?X_1, ?N) \land p_2(?X_2, ?N) \land \text{comp}(?X_1, ?X_2, ?X)$
    - Where $p_i$ is a predicate identifying the graph pattern $P_i$, $g$ is the active graph of dataset $D$.
  - Rules for modeling compatible mappings: (brave join used by SPARQL\textsubscript{WG})
    - $\text{comp}(X,X,X) \leftarrow \text{term}(X)$
    - $\text{comp}(X, \text{null}, X) \leftarrow \text{term}(X)$
    - $\text{comp}(X,X,X) \leftarrow \text{Null}(X)$
    - $\text{comp}(\text{null}, X,X) \leftarrow \text{term}(X)$
SPARQL\textsubscript{C} \iff nr-Datalog\neg

- \text{nr-Datalog}\neg \Rightarrow SPARQL\textsubscript{C}
  - Datalog facts as RDF triples: \( p(c_1, \ldots, c_n) \Rightarrow \{(b, \text{predicate}, p), (b, \text{rdf: 1}, c_1), \ldots, (b, \text{rdf: n}, c_n)\} \)
  - Datalog substitutions as SPARQL mappings
  - Datalog rule as graph patterns:
    \[
    g(L_1 \land \cdots \land L_s \land \neg L_{s+1} \land \cdots \land \neg L_t \land L_1^{eq} \land \cdots \land L_u^{eq}) \Rightarrow \\
    (((\cdots (((g(L_1) \land \cdots \land g(L_s))) \land g(L_{s+1})) \cdots) \land \cdots) \land \cdots) \land g(L_t))
    \\
    \text{FILTER}(L_1^{eq} \land \cdots \land L_u^{eq})
    \]
SPARQL\( \Leftrightarrow \) nr-Datalog\( \neg \)

- Example:
  - Datalog rule: \( Q(\?X, \?N) \leftarrow \text{name}(\?X, \?N) \land \neg \text{email}(\?X, \?E) \)
  - SPARQL queries:
    - SELECT ?X, ?N
    - FROM g
    - WHERE (  
      - ( (?Y predicate name)  
        - AND (?Y rdf: 1 ?X)  
        - AND (?Y rdf: 2 ?N)  
        - MINUS  
        - ( (?Z predicate email)  
          - AND (?Z rdf: 1 ?X)  
          - AND (?Z rdf: 2 ?E)  
        )  
      )
  )
SPARQL $\leftrightarrow$ Relational Algebra

- Since nr-Datalog$^\neg$ and relational algebra have the same expressive power.
From SPARQL to Rules
(and back)

Axel Polleres

WWW2007
Rules are important for the semantic web

Existing approaches with RDF: CWM, SWI-Prolog, RIF

Lots of existing (non-RDF) work to draw from (databases!)

How can SPARQL and rules be combined?
RDF in Datalog

- Use a built-in predicate:
  
  \[ \text{rdf[URL]}(S,P,O) \]

- Basic SPARQL graph patterns are easy:

  ```sparql
  SELECT ?X ?Y
  FROM <http://alice.org>
  FROM <http://ex.org/bob>
  WHERE { ?X a foaf:Person . ?X foaf:name ?Y . }
  ```

  becomes

  ```prolog
  answer1(X,Y,def)  :- triple(X,"rdf:type","foaf:Person",def),
                     triple(X,"foaf:name",Y,def).
  ```
More Complex Patterns

- OPTIONAL and UNION make this tricky. Why?
- Because some variables can end up unbound
- Performing joins on possibly-unbound data is "strange"
Example

SELECT *
FROM ...
WHERE { { ?X1 a foaf:Person . OPTIONAL { ?X1 foaf:name ?N } }
{ ?X2 a foaf:Person . OPTIONAL { ?X2 foaf:nick ?N } } }

<table>
<thead>
<tr>
<th>?X1</th>
<th>?N</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:a</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>_:b</td>
<td></td>
</tr>
<tr>
<td>_:c</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>alice.org#me</td>
<td>&quot;Alice&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X2</th>
<th>?N</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:a</td>
<td></td>
</tr>
<tr>
<td>_:b</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>_:c</td>
<td>&quot;Bobby&quot;</td>
</tr>
<tr>
<td>alice.org#me</td>
<td></td>
</tr>
</tbody>
</table>

- What results should we expect?
3 Notions of Join Compatibility

- Bravely compatible
  - Unbound variables are compatible with everything: (null,null), (T, null), (null, T), (T,T)

- Cautiously compatible
  - Unbound variables are compatible with each other: (null,null), (T,T)

- Strictly compatible
  - Only bound terms can be compatible: (T,T)
## Brave Compatibility

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_:a</td>
<td>&quot;Bob&quot;</td>
<td>_:a</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>_:b</td>
<td>&quot;Bob&quot;</td>
<td>_:b</td>
<td>&quot;Bobby&quot;</td>
</tr>
<tr>
<td>_:c</td>
<td>&quot;Alice&quot;</td>
<td>_:c</td>
<td>alice.org#me</td>
</tr>
<tr>
<td>alice.org#me</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
?X1 & & ?N & & X2 \\
_:a & & "Bob" & & \text{alice.org#me} \\
_:a & & "Bob" & & \text{alice.org#me} \\
_:b & & "Alice" & & \text{alice.org#me} \\
_:b & & "Bobby" & & \text{alice.org#me} \\
_:b & & "Bob" & & \text{alice.org#me} \\
_:c & & "Alice" & & \text{alice.org#me} \\
_:c & & "Bob" & & \text{alice.org#me} \\
_:c & & "Bobby" & & \text{alice.org#me} \\
alice.org#me & & "Alice" & & \text{alice.org#me} \\
alice.org#me & & "Alice" & & \text{alice.org#me} \\
alice.org#me & & "Alice" & & \text{alice.org#me} \\
\end{align*}
\]
Cautious Compatibility

<table>
<thead>
<tr>
<th>?X1</th>
<th>?N</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:_a</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>_:_b</td>
<td>null</td>
</tr>
<tr>
<td>_:_c</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>alice.org#me</td>
<td>&quot;Alice&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X2</th>
<th>?N</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:_a</td>
<td>null</td>
</tr>
<tr>
<td>_:_b</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>_:_c</td>
<td>&quot;Bobby&quot;</td>
</tr>
<tr>
<td>alice.org#me</td>
<td>null</td>
</tr>
</tbody>
</table>

=  

<table>
<thead>
<tr>
<th>?X1</th>
<th>?N</th>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:_b</td>
<td>null</td>
<td>alice.org#me</td>
</tr>
<tr>
<td>_:_b</td>
<td>null</td>
<td>_:_a</td>
</tr>
<tr>
<td>alice.org#me</td>
<td>&quot;Alice&quot;</td>
<td>_:_b</td>
</tr>
</tbody>
</table>
Strict Compatibility

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_ : a</td>
<td>&quot;Bob&quot;</td>
<td>_ : a</td>
<td>NULL</td>
</tr>
<tr>
<td>_ : b</td>
<td>NULL</td>
<td>_ : b</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>_ : c</td>
<td>&quot;Bob&quot;</td>
<td>_ : c</td>
<td>&quot;Bobby&quot;</td>
</tr>
<tr>
<td>alice.org#me</td>
<td>&quot;Alice&quot;</td>
<td>alice.org#me</td>
<td>NULL</td>
</tr>
</tbody>
</table>

\[ ?X1 = \]

<table>
<thead>
<tr>
<th>?X1</th>
<th>?N</th>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice.org#me</td>
<td>&quot;Alice&quot;</td>
<td>_ : b</td>
</tr>
</tbody>
</table>
Standard SPARQL

- "Bravely compatible" is standard SPARQL
- But, cautious/strict compatibility are also interesting:
  - Closer to traditional (relational) semantics
  - Simpler to implement
- SPARQL → Rules translation can use all three with only slight variations
Translating SPARQL to Datalog

- Recursive set of rules for translating SPARQL graph patterns to datalog
- Defined in terms of cautious compatibility
- Extra rules can be added to use brave or strict compatibility
- Potentially exponential program size when dealing with unbound variables
Translation Example

```
SELECT *
FROM ... 
WHERE { { ?X1 a foaf:Person . OPTIONAL { ?X1 foaf:name ?N } 
    { ?X2 a foaf:Person . OPTIONAL { ?X2 foaf:nick ?N } } }

triple(S,P,0,def) :- rdf["ex.org/bob"](S,P,0).
triple(S,P,0,def) :- rdf["alice.org"](S,P,0).

answer1(N,X1,X2,def) :- answer2(N,X1,def), answer4(N,X2,def).

answer2(N, X1,def) :- triple(X1,"a","Person",def),
    triple(X1,"name",N,def).
answer2(null,X1,def) :- triple(X1,"a","Person",def),
    not answer3(X1,def).
answer3(X1,def) :- triple(X1,"name",N,def).

answer4(N, X2,def) :- triple(X2,"a","Person",def),
    triple(X2,"nick",N,def).
answer4(null,X2,def) :- triple(X2,"a","Person",def),
    not answer5(X2,def).
answer5(X2,def) :- triple(X2,"nick",N,def).
```
Translation Example

```prolog
SELECT *
FROM ...
WHERE { { ?X1 a foaf:Person . OPTIONAL { ?X1 foaf:name ?N } 
    { ?X2 a foaf:Person . OPTIONAL { ?X2 foaf:nick ?N } } } }
```

```prolog
triple(S,P,O,def) :- rdf["ex.org/bob"](S,P,O).
triple(S,P,O,def) :- rdf["alice.org"](S,P,O).
answer1(N,X1,X2,def) :- answer2(N,X1,def), answer4(N,X2,def).
answer1(N,X1,X2,def) :- answer2(N,X1,def), answer4(null,X2,def)
answer1(N,X1,X2,def) :- answer2(null,X1,def), answer4(N,X2,def)
answer2(N,  X1,def) :- triple(X1,"a","Person",def),
                     triple(X1,"name",N,def).
answer2(null,X1,def) :- triple(X1,"a","Person",def),
                        not answer3(X1,def).
answer3(X1,def) :- triple(X1,"name",N,def).
answer4(N,  X2,def) :- triple(X2,"a","Person",def),
                     triple(X2,"nick",N,def).
answer4(null,X2,def) :- triple(X2,"a","Person",def),
                       not answer5(X2,def).
answer5(X2,def) :- triple(X2,"nick",N,def).
```
"and back..." - Rules in SPARQL

- CONSTRUCT queries can be thought of as rules that produce new triples
- Similar translation to Datalog as SELECT queries
- Combination of RDF data and CONSTRUCT queries gives powerful recursive rules
Importance of SPARQL ⇔ Rules

- SPARQL as rules allows the use of existing Datalog tools when working with RDF
- CONSTRUCT queries can provide recursive rules with SPARQL syntax
- Can give clear semantics to SPARQL extensions (nested queries, MINUS operator)
- Rules can be a more natural for expressing semantics (constraints, business logic)
Future Directions

- SPARQL $\rightarrow$ Datalog is useful, but Datalog $\rightarrow$ SPARQL leaves much to be desired:

$$Q(?N) :- \text{name}(?X,?N) \Rightarrow$$

SELECT ?N WHERE {
  ?y :predicate <name> ;
  rdf:_1 ?X ;
  rdf:_2 ?N .
}

- Nobody writes RDF this way!
Future Directions

• Instead, we want something like this:
  
  \[
  \text{SELECT } ?N \text{ WHERE } \{
  \begin{align*}
  ?X :\text{name} &\ ?N .
  \end{align*}
  \}
  \]

• ... suggesting the need for a way to map between n-ary atoms and triple(pattern)s

• Then SPARQL and Datalog could be used interchangeably
Questions?