CycQL: A SPARQL Adapter for OpenCyc

Steve Battle

Sysemia Ltd, Bristol & Bath Science Park,
Dirac Crescent, Emerson's Green, Bristol BS16 7FR, UK
steve.battle@sysemia.co.uk,
WWW home page: http://www.sysemia.co.uk

Abstract. CycQL is an Apache Jena/ARQ based SPARQL adapter for OpenCyc 4.0. It enables the Cyc inference engine to be used with Semantic Web tools via a SPARQL endpoint. Cyc achieves scalability and optimizes inference by restricting the search space to a relevant subset of microtheories. With this adapter, Cyc microtheories are identified with RDF named graphs. This paper demonstrates how greater efficiency is achieved by maximizing the chunks of SPARQL algebra that are translated into CycL.

Keywords: Cyc, OpenCyc, CycL, Jena, RDF, SPARQL

1 Introduction

Cyc [1] was one of the first Artificial Intelligence systems to develop an ontological approach to organizing knowledge. The modularity this affords enables Cyc to reason scalably in highly complex domains. OpenCyc 4.0, released in June 2012, includes the full Cyc ontology. However, Cyc is often overlooked as a reasoner on the web because of a lack of integration points with other semantic technologies. The full opencyc ontology is available as a downloadable OWL (Web Ontology Language) file, and individual concepts may be downloaded in RDF from the OpenCyc website. In addition, UMBEL [2] identifies a subset of OpenCyc which can be used as an upper ontology for the purpose of ontology alignment.

This paper describes the development of a SPARQL adapter for OpenCyc, known as CycQL [3], which enables a Cyc instance to support a SPARQL endpoint. The ARQ SPARQL evaluator is an extensible framework for implementing SPARQL adapters. ARQ parses the query into SPARQL algebra; a tree structure representing query operators and triple or quad patterns. A query evaluation walks this algebra and combines the resulting bindings to produce the final result set.

The CycQL adapter comprises a number of components. Firstly, a wrapper for the CycAccess object implements the Jena/ARQ DatasetGraph, providing an RDF graph representation of the Cyc knowledge base. Secondly, each step of query evaluation is routed through a custom OpExecutor that determines whether a given operator in the SPARQL abstract syntax tree can be compiled directly into CycL, or should be executed as usual by ARQ. Thirdly, SPARQL

operators and patterns to be compiled into CycL are routed by the OpExecutor to a custom StageGenerator that generates bindings for a given *stage* in the SPARQL algebra. This paper explores the compilation of ever larger stages of the query to create a more efficient adapter.

2 Mapping CycL to RDF

The CycL representation is a predicate-calculus like language that allows us to assert facts about the world, and to express queries over those facts. A CycL expression is a bracketed list with the predicate at the head of that list. It may contain constants, prefixed with `#$`. For example, the fact that Pluto orbits the Sun could be stated as follows.

```
(#$orbits #$PlanetPluto #$TheSun)
```

Translating this binary expression into RDF (Resource Description Framework) [4] involves transposing the order of the terms to give the typical subject, predicate, object arrangement. It also involves assigning URIs to the Cyc constants. In this work, the CycAccess object provided by OpenCyc is wrapped as an Apache Jena/ARQ DatasetGraph supporting a graph representation of an RDF Dataset. A URI base can be set on this wrapper which is simply prepended to the constant value to generate a URI. For example, the base may be set to the OpenCyc namespace `<http://sw.opencyc.org/2012/05/10/concept/en/>`, which may be used for English language versions of concepts. Alternatively, the OpenCyc corpus may supply an *rdfURI* property and in these cases this supplied URI is used in preference to the generated URI. The above assertion may be written (in RDF Turtle) as follows, with all URIs relativized to the document base.

```
@base <http://sw.opencyc.org/2012/05/10/concept/en/> .
<PlanetPluto> <orbits> <TheSun> .
```

Type information is asserted in CycL with the `#$isa` predicate, which maps directly to `rdf:type`. Following Pluto's demotion to a dwarf-planet in 2006, we may make the following assertion.

```
(#$isa #$PlanetPluto #$DwarfPlanet)
```

Using Turtle short-hand for `rdf:type`, the above may be written as follows, again with individuals expressed as document relative URIs (assuming that the `@base` is set as above).

```
<PlanetPluto> a <DwarfPlanet> .
```

The constant `#$PlanetPluto` is described as being atomic, as it has no internal structure, simply properties that are extrinsic to that atomic concept. Non-Atomic Terms (NATs) are expressed as lists and do not have a truth value as such, but can be used within expressions. These may be used to express

functional relationships between objects. Non-Atomic Terms can also be used to express qualifying information about a value, such as its type. For example, the orbital period of Pluto can be expressed as a number qualified by the units it is measured in.

```
(#$orbitalPeriod #$PlanetPluto (#$DaysDuration 90739))
```

Such unary, un-nested NATs map nicely to RDF custom datatypes. The above may be expressed in Turtle as follows.

```
<PlanetPluto> <orbitalPeriod> "90739"^^<DaysDuration> .
```

CycL is able to form complex logical expressions using the *truth functions* (`#$and`, `#$or`, `#$not`, `#$implies`). Pluto has five known moons, and If we are to believe recent online polls, the recently discovered moons, *P4* (2011) and *P5* (2012), are to be named 'Vulcan' and 'Cerberus'. This knowledge can be asserted in CycL as a logical conjunction.

```
(#$and
  (#$orbits #$Charon-MoonOfPluto #$PlanetPluto)
  (#$orbits #$Nix-MoonOfPluto #$PlanetPluto)
  (#$orbits #$Hydra-MoonOfPluto #$PlanetPluto)
  (#$orbits #$Vulcan-MoonOfPluto #$PlanetPluto)
  (#$orbits #$Cerberus-MoonOfPluto #$PlanetPluto) )
```

In RDF it is straightforward to represent a conjunction of triples such as these, but we cannot directly represent disjunction, negation, or implication. This highlights the utility of using highly expressive languages such as CycL alongside RDF. We see later on how this expressiveness enables us to define CycL rules that may be used in conjunction with a SPARQL query.

2.1 Limitations of the mapping

The mapping currently supports Non-Atomic Terms that are functions of literal values. The mapping does not currently support NATs that are functions of a constant. In such cases the function name typically ends with 'Fn'. For example, we could talk about the *moons of Pluto* using the following CycL NAT.

```
(#$MoonFn #$PlanetPluto)
```

This doesn't assert anything by itself but could be used within an assertion such as the following.

```
(#$isa #$Charon-MoonOfPluto (#$MoonFn #$PlanetPluto))
```

As long as the function is unary, it can be expressed in RDF as a pair of triples, where `moonofpluto` is a so-called blank node; the object of a functional relationship `MoonFn`. The result is known as a Non-Atomic Reified Term and may be represented in RDF as follows.

```
<PlanetPluto> <MoonFn> _:moonofpluto .
<Charon-MoonOfPluto> a _:moonofpluto .
```

One of the advantages of CycL is that it can express n-ary relationships. Rather than consider complex mappings of n-ary (for $n > 2$) expressions into RDF, these are instead invisible to the RDF mapping. It is assumed that Cyc inferencing may be used to unpack any such expression to generate the equivalent form in terms of binary predicates, if required. This can be done on demand; effectively defining a *magic* (or computed) property.
An example of this, which will be explained later on, is the use of an `orbitalRadius` property of a planet which should be computed (on demand), but we can treat as though it were asserted as a simple relationship.

```
<PlanetPluto> <orbitalPeriod> "90739"^^<DaysDuration> .
```

3 Mapping SPARQL to CycL queries

SPARQL (SPARQL Protocol and RDF Query Language) [5] is a widely used query language for RDF datasets. The following SPARQL query selects for dwarf planets orbiting the Sun together with their moons, if any. The results are shown below.

```
PREFIX : <>
SELECT ?planet ?moon
WHERE {
  ?planet a :DwarfPlanet ; :orbits :TheSun
  OPTIONAL { ?moon :orbits ?planet }
}
```

```
-----------------------------------------
| planet       | moon                   |
=========================================
| :PlanetPluto | :Charon-MoonOfPluto    |
| :PlanetPluto | :Nix-MoonOfPluto       |
| :PlanetPluto | :Hydra-MoonOfPluto     |
| :PlanetPluto | :Vulcan-MoonOfPluto    |
| :PlanetPluto | :Cerberus-MoonOfPluto  |
-----------------------------------------
```

The query above contains a number of triple patterns that can be translated into CycL as were the basic axioms of our planetary system. As in SPARQL, CycL represents variables by prefixing them with a '?'. The three triple patterns that appear above may therefore be individually translated into CycL as follows.

```
(#$isa ?PLANET #$DwarfPlanet)
(#$orbits ?PLANET #$TheSun)
(#$orbits ?MOON ?PLANET)
```

3.1 Microtheories as graphs

The Cyc knowledge-base is divided into a number of microtheories, each of which corresponds to a particular domain of knowledge. Cyc's knowledge of the planets is mostly held in `UniverseDataMt`. Microtheories are hierarchically arranged so that facts from any one microtheory can be inherited by more specialized microtheories. Microtheories are identified by a constant (microtheory names typically include 'Mt'), so they can be treated like any other individual and be assigned a URI.
Each access to the Cyc knowledge base must define a specific microtheory. SPARQL defines a *default* graph against which Basic Graph Patterns (BGPs), the sets of triples that appear in a query, are matched. The initial setting for this default graph is set in the DatasetGraph, but can be overriden with the addition of a `FROM` clause in the query. The query below explicitly defines a microtheory to be used as the new default graph.

```
PREFIX : <>
SELECT ?planet
FROM :CurrentWorldDataCollectorMt-NonHomocentric
WHERE {
  ?planet a :Planet ; :orbits :TheSun
}
```

Alternatively, to allow knowledge to be integrated from multiple sources graphs may be explicitly named within the body of the query using the `GRAPH` clause. Knowledge about dwarf planets appears in a a more general microtheory than planets and their orbits, so it can be more efficient to target specific parts of the query at specific microtheories as in the example below.

```
PREFIX : <>
SELECT ?planet
FROM :CurrentWorldDataCollectorMt-NonHomocentric
FROM NAMED :UniverseDataMt
WHERE {
  ?planet a :DwarfPlanet
  GRAPH :UniverseDataMt {
    ?planet :orbits :TheSun
  }
}
```

3.2 Negation

Negation is not available in the RDF representation, though it is available as a feature in SPARQL query. While the query at the beginning of this section selects *for* dwarf planets, this time we wish to filter them out using the SPARQL 1.1 `NOT EXISTS` clause.

```
PREFIX nat: <java:org.opencyc.sparql.function.nat.>
PREFIX : <>
SELECT ?planet ?orbital_period
FROM :CurrentWorldDataCollectorMt-NonHomocentric
WHERE {
  ?planet a :Planet ;
    :orbits :TheSun ; :orbitalPeriod ?orbital_period
  FILTER (NOT EXISTS { ?planet a :DwarfPlanet })
}
ORDER BY nat:Integer(?orbital_period)
```

In this example, we make use of Johannes Kepler's observation that the distance of a planet from the Sun is proportional to its orbital period. More on this later, but for now this knowledge can be used to define a natural ordering over the planets as seen in the results below.

```
-------------------------------------------
| planet         | orbital_period         |
===========================================
| :PlanetMercury | "88"^^:DaysDuration    |
| :PlanetVenus   | "225"^^:DaysDuration   |
| :PlanetEarth   | "365"^^:DaysDuration   |
| :PlanetMars    | "687"^^:DaysDuration   |
| :PlanetJupiter | "4329"^^:DaysDuration  |
| :PlanetSaturn  | "10753"^^:DaysDuration |
| :PlanetUranus  | "30660"^^:DaysDuration |
| :PlanetNeptune | "60152"^^:DaysDuration |
-------------------------------------------
```

We see the Non-Atomic Term representing the orbital period in the output but the application of the function `nat:Integer` has yet to be explained. As these NATs are not numbers, any attempt to `ORDER BY` the term directly would be based on the order of their lexical value alone. The first character would be the most significant and would have the undesirable effect of placing Mercury at the outer edge of the solar system. The application of a custom java function in the *nat* namespace casts this lexical value to the indicated type, in this case an (XSD) Integer.

3.3 CycL Compilation levels

Apache Jena/ARQ defines a StageGenerator interface for executing triple patterns, quad patterns and other algebraic operations and returns a binding iterator. These patterns and operators are compiled into a CycL query to be evaluated by Cyc.
There are three natural levels at which one may group content from the original query for translation into CycL. These levels are enumerated below, each building on, and having greater efficiency than, the previous level.

The *negation* query of the previous subsection will be used to demonstrate this compilation into CycL at different levels of grouping, with each level offering greater efficiency. These efficiencies are gained by reducing the depth of the search tree from a depth of 4 using the triple-pattern level, down to 1 with the operator-composition level.

triple-pattern level At the simplest level, a query can be decomposed into individual triple-patterns. The four steps below represent a single branch of the search tree (of depth 4), with `?PLANET` bound in step 1 to `#$PlanetEarth`.

1. `(#$isa ?PLANET #$Planet)`
2. `(#$orbits #$PlanetEarth #$TheSun)`
3. `(#$orbitalPeriod #$PlanetEarth ?ORBITAL-PERIOD)`
4. `(#$isa #$PlanetEarth #$DwarfPlanet)`

Note that, in this case, the evaluation of `NOT EXISTS` takes place in the SPARQL engine rather than Cyc. If the final query returns no results then it succeeds.

graph-pattern level SPARQL queries include blocks of triples that are known as Basic Graph Patterns (BGPs), or where named graphs are introduced, as Named Graph Patterns (NGPs). A pattern containing multiple triples may be translated by the StageGenerator into a single logical conjunction to be evaluated by Cyc in a single step. The depth of the search tree is thereby reduced to 2.

1. ```
   (#$and
         (#$isa ?PLANET #$Planet)
         (#$orbits ?PLANET #$TheSun)
         (#$orbitalPeriod ?PLANET ?ORBITAL-PERIOD))
   ```
2. `(#$isa #$PlanetEarth #$DwarfPlanet)`

As above, the evaluation of `NOT EXISTS` takes place in the SPARQL engine.

operator-composition level Each step of query evaluation is routed through an OpExecutor that evaluates graph patterns, filters, sequences and joins. The OpExecutor composes the largest units of the SPARQL algebra that make sense as a single CycL query. The composed units are then routed to the StageGenerator to generate result bindings. The depth of this search is just 1.

1. ```
   (#$and
      (#$isa ?PLANET #$Planet)
      (#$not (#$isa ?PLANET #$DwarfPlanet))
      (#$orbits ?PLANET #$TheSun)
      (#$orbitalPeriod ?PLANET ?ORBITAL-PERIOD))
   ```

Note that in this case, Cyc evaluates the negation directly using `#$not`.

4 Functions

We wish to compute which planets lie in the *Circumstellar Habitable Zone* (CHZ). This is the famous 'Goldilocks zone' containing the Earth and Mars, defined to be between 0.725 and 3 Astronomical Units from the Sun.
Cyc defines a number of functions, which can be expressed as CycL Non-Atomic Terms. These functions are non-reifiable, meaning that they do not represent individuals in their own right, but are instead intended to be evaluated. We will make use of the mathematical functions, `#$ExponentFn` and `#$QuotientFn`. In the first instance we show how these functions can be surfaced within an Apache Jena/ARQ `LET` statement. This binds the variable on the left-hand side to the value of the expression on the right-hand side. The *cyc* namespace is introduced as a way to identify Cyc functions. Access to Cyc functions is provided by registering them with an ARQ function registry. Given the function URI, the function factory queries Cyc for the arity and return type of the function. In this case the return type is a double, so this value (`?AU`) may be used directly in the `ORDER BY` clause.
Kepler's third law states that "The square of the orbital period of a planet is directly proportional to the cube of the semi-major axis of its orbit," $P^2 = R^3$, where P is expressed in years, and R in Astronomical Units (AUs).

```
PREFIX cyc: <http://www.opencyc.org#>
PREFIX nat: <java:org.opencyc.sparql.function.nat.>
PREFIX : <>
SELECT ?planet ?AUs
FROM :UniverseDataMt
WHERE {
  ?planet a :Planet ;
    :orbits :TheSun ; :orbitalPeriod ?orbital_period
    LET (?AU := cyc:ExponentFn(
      cyc:QuotientFn(nat:Double(?orbital_period),365),
      cyc:QuotientFn(2,3)) )
    FILTER (0.725 < ?AU && ?AU < 3.0)
} ORDER BY ?AU
```

This query is translated into the CycL below, with only the remaining ORDER BY clause being evaluated by the SPARQL adapter. Note the introduction of the equality to extract the lexical value (?VAR1) of the orbital period NAT (??VAR0 binds to the name of the NAT). The results follow.

```
(#$and
  (#$isa ?PLANET #$Planet)
  (#$orbits ?PLANET #$TheSun)
  (#$orbitalPeriod ?PLANET ?ORBITAL-PERIOD)
  (#$equals ?ORBITAL-PERIOD (??VAR0 ?VAR1))
  (#$evaluate ?AU (#$ExponentFn (#$QuotientFn ?VAR1 365) (#$QuotientFn 2 3)))
  (#$lessThan 0.725 ?AU)(#$lessThan ?AU 3.0) )
```

```
-------------------------------------------------
| planet       | AU                             |
=================================================
| :PlanetEarth | "1.0"^^xsd:double              |
| :PlanetMars  | "1.5244361831950344"^^xsd:double |
-------------------------------------------------
```

5 Rules

Although the approach described in the previous section returns the correct results, it complicates matters with the unnecessary inclusion of Kepler's law within the body of the query itself.
In this section we will develop a CycL rule to perform a simple backward-chaining inference triggered by a SPARQL query. Cyc may, of course, perform other kinds of inference such as forward chaining, triggered *opportunistically* by the initial assertion of the axioms. At this point it may, for example, perform type closure over `#$isa` relationships.
The aim is to use Cyc rules alongside SPARQL. These backward chaining rules are only invoked at the time the query is made. Neither are they derived from the query; they are native Cyc rules. While various schemes exist to implement rules in SPARQL they tend to be forward chaining engines. One example of this is the use of chained SPARQL `CONSTRUCT` queries as found in the SPARQL Inferencing Notation (SPIN) [6].
We define Kepler's law using a CycL rule as follows. The orbitalRadius is inferred from the orbital period using Kepler's 3rd law.

```
(#$implies
  (#$and
    (#$orbitalPeriod ?BODY (#$DaysDuration ?PERIOD))
    (#$evaluate ?RADIUS
      (#$ExponentFn (#$QuotientFn ?PERIOD 365)(#$QuotientFn 2 3))))
  (#$orbitalRadius ?BODY (#$AstronomicalUnits ?RADIUS)))
```

The SPARQL query is simplified using the magic property, `#$orbitalRadius`.

```
PREFIX nat: <java:org.opencyc.sparql.function.nat.>
PREFIX : <>
SELECT ?planet ?orbital_radius
FROM :UniverseDataMt
WHERE {
  ?planet a :Planet ;
    :orbits :TheSun ; :orbitalRadius ?orbital_radius
  FILTER (0.725 < nat:Double(?orbital_radius) && nat:Double(?orbital_radius) < 3.0)
}
ORDER BY nat:Double(?orbital_radius)
```

The query above is translated into the CycL below using operator-composition to combine the Basic Graph Pattern and the FILTER clause. When Cyc evaluates the goal with the predicate `#$orbitalRadius`, it triggers the rule above to compute the value.

```
(#$and
    (#$isa ?PLANET #$Planet)
    (#$orbits ?PLANET #$TheSun)
    (#$orbitalRadius ?PLANET ?ORBITAL-RADIUS)
    (#$equals ?ORBITAL-RADIUS (??VAR0 ?VAR1))
    (#$lessThan 0.725 ?VAR1)
    (#$lessThan ?VAR1 3) )
```

6 Conclusion and next steps

CycQL is a useful addition to the tool-box available to users of OpenCyc. As a proof-of-concept it demonstrates the efficiencies to be gained by compiling as much as the query as possible into CycL before handing off to Cyc. The next steps include refining the RDF mapping to support NARTs (Non-Atomic Reified Terms), and to provide a more complete coverage of SPARQL. This will include the addition of a *describe-handler* supporting SPARQL `DESCRIBE` queries, and support for multiple `FROM` clauses in the SPARQL query.

References

1. Lenat, Douglas. B., Guha, R. V.: Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project (1989)
2. Bergman, M.K., UMBEL: A Subject Concepts Reference Layer for the Web, <http://www.slideshare.net/mkbergman/umbel-a-subject-concepts-reference-layer-for-the-web>
3. CycQL SPARQL adapter for OpenCyc, <http://code.google.com/p/gloze/wiki/CycQL>.
4. Klyne, G., Carroll, J.J., Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, 10 February 2004, <http://www.w3.org/TR/rdf-concepts/>.
5. Harris, S., Seaborne, A., SPARQL 1.1 Query Language: W3C Proposed Recommendation, 08 November 2012, <http://www.w3.org/TR/sparql11-query/>.
6. Knublauch, H., Hendler, J., Idehen, K., SPIN - Overview and Motivation, W3C Member Submission 22 February 2011, <http://www.w3.org/Submission/spin-overview/>.