BELQL Documentation

Release 0.0.1

Charles Tapley Hoyt

Apr 12, 2018

Contents

BEL Query Language (BELQL) is a set of commands inspired by SQL, Cypher, and SPARQL to make reproducible queries over networks encoded in the Biological Expression Language (BEL).

CHAPTER 1

Inspiration from the BEL Commons Query Builder

The assembly step consists of combining several networks. Because they are all encoded in BEL, they follow the same schema and no specific joins need to be mentioned.

```
SELECT EDGES
FROM "Network 1" AND "Network 2" [AND ...]
```

Additionally, the query builder applies seed methods. These can be listed and each have their own sub-language for how the arguments are interpreted. For example, seeding by subgraphs can be written as a list of strings in quotes.

```
SELECT EDGES
FROM "Network 1" AND "Network 2"
SEED ANNOTATIONS "Subgraph:Insulin Subgraph" OR "Subgraph:DKK1 Subgraph"
```

Since seed methods are implicitly combine, several can be used in succession.

```
SELECT EDGES
FROM "Network 1" AND "Network 2"
SEED ANNOTATIONS "Subgraph:Insulin Subgraph" OR "Subgraph:DKK1 Subgraph"
SEED NEIGHBORS p(HGNC:APP) OR p(HGNC:BACE1) OR p(HGNC:BACE2)
```

Finally, after seeding, pipeline functions can be applied that come from PyBEL Tools. This can be written in natural language that will be normalized and the function is looked up.

```
SELECT EDGES
FROM "Network 1" AND "Network 2"
SEED ANNOTATIONS "Subgraph:Insulin Subgraph" OR "Subgraph:DKK1 Subgraph"
SEED NEIGHBORS p(HGNC:APP) OR p(HGNC:BACE1) OR p(HGNC:BACE2)
APPLY Infer Central Dogma
APPLY Prune Central Dogma
```

Some pipeline functions require BEL nodes as argument, and they can be written directly after a colon:

```
SELECT EDGES
FROM "Network 1" AND "Network 2"
```

```
SEED ANNOTATIONS "Subgraph:Insulin Subgraph" OR "Subgraph:DKK1 Subgraph"
SEED NEIGHBORS p(HGNC:APP) OR p(HGNC:BACE1) OR p(HGNC:BACE2)
APPLY Infer Central Dogma
APPLY Prune Central Dogma
APPLY Expand by Neighborhood: p(HGNC:DKK1)
APPLY Delete node: p(HGNC:APP)
```

CHAPTER 2

What sorts of things can be selected from a query language?

In SQL, the columns from a table or multiple joined tables are selected and returned as a table. Similarly in SPARQL, a subset of the variables that have been used in the query (whether they appear in one or more subjects, predicates, or objects) can be selected to be returned as a table once for each combination in which they appear that match the query.

BEL has the advantage of being a network-based format, so there are multiple sorts of things people might want to select while querying it. Most obviously is to return a list of BEL edges, which can be displayed as a network. Further, a list of edges can be summarized as a list of nodes appearing in either their respective subjects or objects.

`SELECT EDGES` results in the edges being returned as a list, which exactly specifies a network. It would also be possible to return this in a tabular format by adding `AS` to the syntax like `SELECT EDGES AS TSV` that would look up the relevant I/O function from PyBEL and stream the results. As a default, Node-Link JSON would be a good choice.

`SELECT NODES` would return a list of the nodes. As a default, Node-Link JSON (without the links) would be appropriate.

CHAPTER 3

Selecting Pathways

It would be even more powerful to select certain types of paths, like finding proteins that increase proteins. This can be encoded as a "meta-path" like `p increases p`. Several metapaths can be combine in succession like `p increases p decreases m`. Such a system can be readily mapped to functions already existing in the PyBEL and related packages.

While this is rudimentary, it would be possible to be more general query systems inspired by SPARQL that allow for arbitrary branching and mapping

```
p(?a) increases p(?b)
p(?b) decreases p(?c)
p(?b) decreases m(?d)
```

These queries would follow the selection of edges with a `WHERE` statement inspired by SPARQL.

```
SELECT EDGES
FROM "Network 1" AND "Network 2"
SEED ANNOTATIONS "Subgraph:Insulin Subgraph" OR "Subgraph:DKK1 Subgraph"
SEED NEIGHBORS p(HGNC:APP) OR p(HGNC:BACE1) OR p(HGNC:BACE2)
APPLY Infer Central Dogma
APPLY Prune Central Dogma
APPLY Expand by Neighborhood: p(HGNC:DKK1)
APPLY Delete node: p(HGNC:APP)
WHERE {
    p(?a) increases p(?b)
    p(?b) decreases p(?c)
    p(?b) decreases m(?d)
}
```

Writing arbitrary BEL with wildcards/variables in the place of functions or names would also be interesting, it would require an entirely new set of functions to be written.

```
SELECT EDGES
FROM "Network 1" AND "Network 2"
SEED ANNOTATIONS "Subgraph:Insulin Subgraph" OR "Subgraph:DKK1 Subgraph"
SEED NEIGHBORS p(HGNC:APP) OR p(HGNC:BACE1) OR p(HGNC:BACE2)
```

```
APPLY Infer Central Dogma
APPLY Prune Central Dogma
APPLY Expand by Neighborhood: p(HGNC:DKK1)
APPLY Delete node: p(HGNC:APP)
WHERE {
    act(p(?a)), ma(kin)) increases p(?b, pmod(?m))
    p(?b, pmod(?m)) decreases p(?c)
}
```

Annotation filters will require some additions of non-BEL-like syntax. One inspiration could come from the Cypher query language, where the attributes of an edge can be checked in brackets like:

```
act(p(?a)), ma(kin)) increases[Species=9606] p(?b, pmod(?m))
```

Or if there are multiple:

```
act(p(?a)), ma(kin)) increases[Species=9606, Species=11090] p(?b, pmod(?m))
```