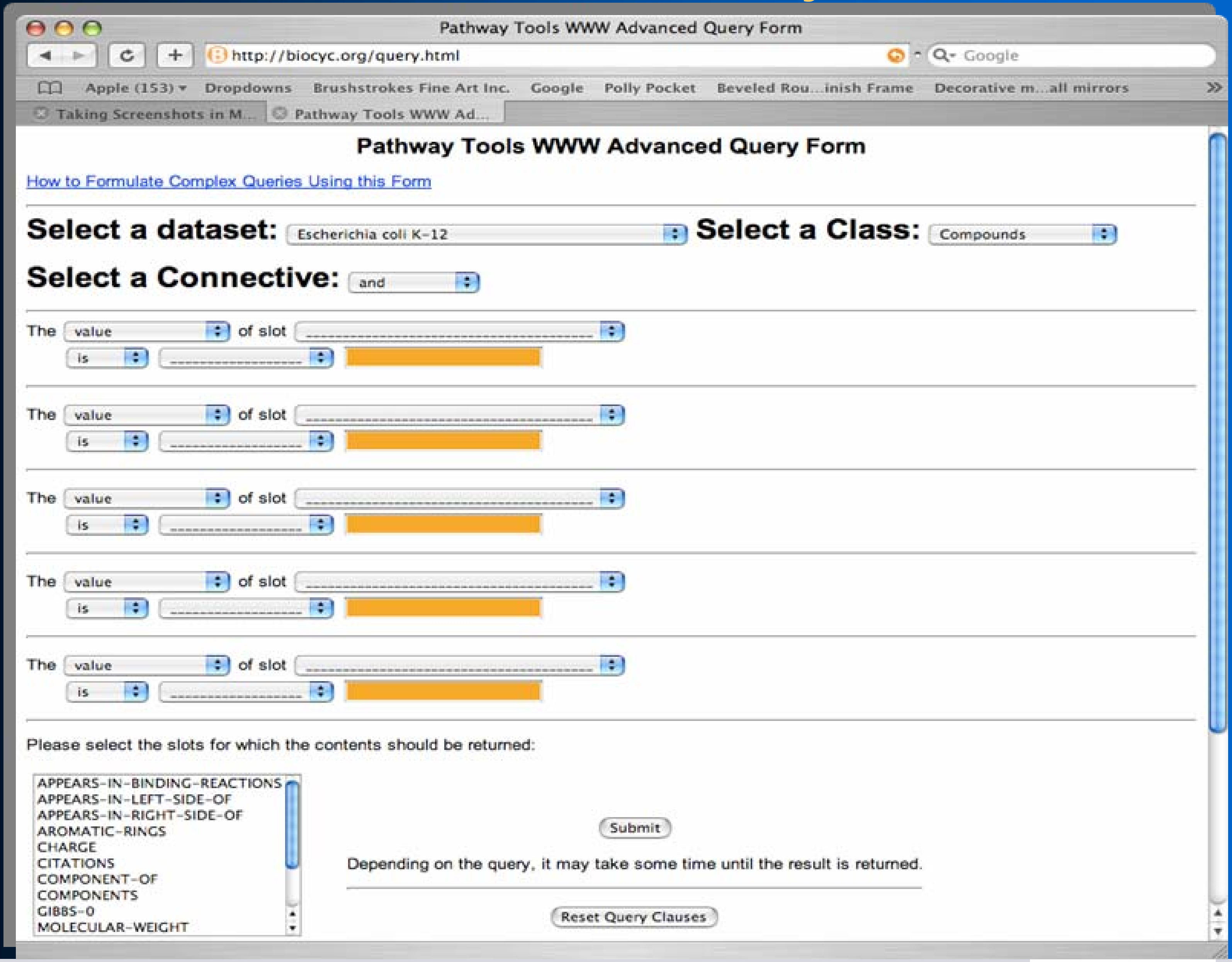
A New Advanced Query Web Page

and its query language

To replace the advanced query web form on www.BioCyc.org

Mario Latendresse
Bioinformatics Research Group
SRI International
Mario@ai.sri.com

The Actual Advanced Query Form


Pathway Tools WWW Advanced Query Form
http://biocyc.org/query.html
Google
Apple (153) Dropdowns Brushstrokes Fine Art Inc. Google Polly Pocket Beveled Rou...inish Frame Decorative m...all mirrors
Taking Screenshots in M... Pathway Tools WWW Ad...

Pathway Tools WWW Advanced Query Form

How to Formulate Complex Queries Using this Form

Select a dataset: Escherichia coli K-12 Select a Class: Compounds

Select a Connective: and

The value of slot
is

The value of slot
is

The value of slot
is

The value of slot
is

The value of slot
is

Please select the slots for which the contents should be returned:

APPEARS-IN-BINDING-REACTIONS
APPEARS-IN-LEFT-SIDE-OF
APPEARS-IN-RIGHT-SIDE-OF
AROMATIC-RINGS
CHARGE
CITATIONS
COMPONENT-OF
COMPONENTS
GIBBS-0
MOLECULAR-WEIGHT

Submit

Depending on the query, it may take some time until the result is returned.

Reset Query Clauses

Major Limitation of the Actual Advanced Query Form

Only one class of objects allowed in one query: cannot join several smaller queries covering different domains.

That is, the search space is done on one datatype: pathway, gene, reaction, protein, or transcription unit, etc.

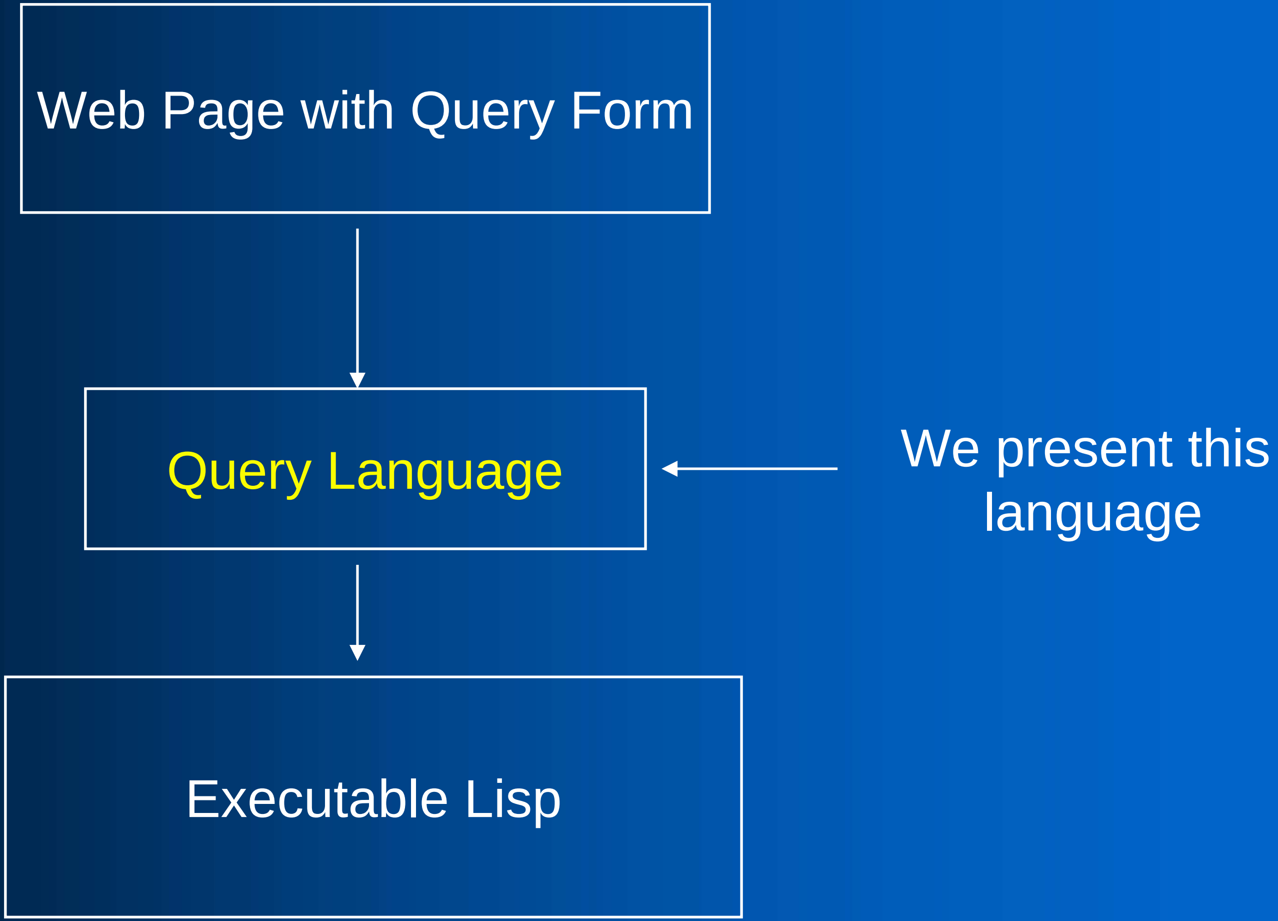
Imagine that you have a list of all pathways of *E. Coli* with their attributes: you then go through each one verifying the conditions on a selected number of attributes.

But you cannot go look into another list, say reactions, while you are looking through pathways.

Other Limitations of the Actual Query Form

1. **Only one global logical connective is allowed in one query: either "and", "or", or "exclusive-or".**
2. **Only one database can be selected in one query.**
3. **Previous queries cannot be combined to form a new query.**
4. **Number of conditionals limited to five.**
5. **The result returned limited to one type of objects.**

Query Language as an intermediate Language



More flexibility is better

In SQL, there are many syntactical and semantics restrictions.

In Lisp, you have a lot of flexibility: no type declaration, many datatypes, etc., but an uncommon syntax.

It becomes easier when there are many ways to express a search.

Purely functional approach: it is easier to reuse (embed) queries.

Our Query Language:

- is the intermediate language used by the new advanced query page;
- has a succinct and readable infix (non-Lisp) syntax to be used directly to write queries;
- will be directly accessible from Lisp: you can type the queries at the command prompt or as any Lisp expression;
- is based on Set Comprehension

Iterates through the elements of sets keeping the ones satisfying some conditionals.

Set Comprehension in Mathematics

- One domain (integers) and one conditional:

  $\{ x : x \text{ in } N, x*x < 100 \}$

  This set is $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- All points on a circle of radius d:

  $\{ (x, y) : x \text{ in } R, y \text{ in } R, x*x + y*y = d*d \}$

- All points inside a sphere of radius d:

  $\{ (x, y, z) : x \text{ in } R, y \text{ in } R, z \text{ in } R, x*x+y*y+z*z < d*d \}$

General Set Comprehension

- { head expression:

  generator, conditional, …, conditional,

  generator, conditional, …, conditional,

  … }

A generator allows the user to "talk about" the elements from a set, and the conditionals select the elements you really want to "talk about". The conditionals used attributes from the elements of this or previous generators.

The head expression is the result to keep for each tuple of elements satisfying the conditionals.

General Set Comprehension

- The generators and conditionals are interpreted from left to right.
- The head expression specifies what to return, the content of the output:

  Often it is simply the selected elements from one generator. Sometimes it is a list of elements from different generators. Sometimes even more complicated…

In Set Comprehension the Search Space

Is the Cartesian products of several datatypes:

Say, while you search through pathways, you can look into the list of reactions, proteins, genes, etc.

The search space can be something like:

(Pathways, Reactions, Proteins, Genes)

A first simple example for PGDBs

{ x : x <- Ecoli^^Reactions, #x^Left < #x^Right }

This is the set of reactions in *E. Coli* **for which the number of left substrates is less than the number of right substrates.**

The generator is Ecoli^^Reactions: the list of reactions from *E. Coli*. There is one conditional.

A Second Simple Example

All reactions of *E. Coli* that are in at least two pathways:

```
{ r : r <- ecoli^^reactions, #r^in-pathway > 1 }
```

It is important to know well the underlying schema of the PGDB to write queries.

Set of Tuples: the output is a table

```
{ (x, #x^Left, #x^Right) :
   x <- Ecoli^^Reactions, #x^Left < #x^Right }
```

The output is a table where each row is one tuple found: each column is an element of the tuple.

In this case, a three columns table where the second and third column are integers: the number of substrates on the left and right.

Two generators: the second is based on each element from the first

Returns all reactions from *E. Coli* that have at least one left substrate in its right side.

{ r : r <- ecoli^^reactions, c <- r^left, c in r^right }

The following will give the same reactions with a different repeated substrate since it also returns the substrate with the reaction:

{ (r,c) : r <- ecoli^^reactions, c <- r^left, c in r^right }

A Variation on Two Generators

Keep the intersection:

```
{ (r, lr) : r <- ecoli^^reactions, lr := r^left ** r^right,
          #lr > 0 }
```

It returns the reactions and the intersection of the left and right substrates for each reaction.

The output is a table where each row has a different reaction: the first column is the reaction and the second column a list of common left/right substrates.

All enzymes of E. coli that catalyze at least two reactions in the same pathway

- { e : e <- ecoli^^proteins, r1 <- e^catalyze,
  r2 <- e^catalyze, r1 != r2,
  #(r1^in-pathway ** r2^in-pathway) > 0 }

A second solution

- { e : p <- ecoli^^pathways, r1 <- p^reactions,
  r2 <- p^reactions, r1 != r2,
  e := r1^enzymatic-reaction,
  e = r2^enzymatic-reaction }

A Third solution

```
{ e : r1 <- ecoli^^reactions, r2 <- ecoli^^reactions,
      r1 != r2, #(r1^in-pathway ** r2^in-pathway) > 0,
      e := r1^enzymatic-reaction^enzyme }
```

Most likely the least efficient: there are over one million pairs (r1, r2) to go through.

Set Comprehension versus List Comprehension

- **A List Comprehension uses lists instead of sets.**
- **A list is useful if:**

  i) Order of elements is important (e.g. attribute reaction-list); or

  ii) Repetition of elements is significant.

- **List Comprehension uses '[…]' instead of '{…}'.**
- **List Comprehension is more efficient to compute.**

  If it is known that no repetition can occur when selecting objects from a database, it is then more efficient to use […]

Limitations of this Query Language

- **In general, recursion is not available: some algorithms you can write in Lisp cannot be written in this query language.**
- **Example, going through tree structures (e.g., complex proteins) with an unknown depth.**

Solutions to these limitations

- Partial solution: adding specific functions like genes-of-reaction, etc. This is nice and simple for the user. (caveat: formal definitions should be provided for these.)
- General solution: adding an escape mechanism to go into Lisp. It is rather simple, but quite messy and complicated.

The (still under design) Web Interface

- **Will keep previous queries available.**
- **Will provide a step by step construction of a query with feedback.**
- **Uses a syntax which is closer to English.**
- **Mostly based on selectable databases, classes, attributes, operators, etc.**

Example Interface, step by step construction with feedback

a) Select object type and database:
 [select type of objects] from [select database]
 constraints:
 [op] [select attribute of object a] [op]
 …
b) Select object type and database:
 [select type of objects] from [select database]
 constraints:
 [op] [select attribute of object] [a,b] [op]
 …
c) …
…

Send in Your Queries!

I would like to know the kind of queries you would like to do on BioCyc

Send in your queries, in English (not Lisp!)

(Or French) to me:

mario@ai.sri.com

I'll acknowledge queries in the documentation.

Acknowledgments

- **Bioinformatics Research Group**
- **The curators who provided some queries, in particular:**
  - Ingrid Keseler
  - Michelle Green