Logic Programming
Applications

Michael Genesereth
Computer Science Department
Stanford University

Applications

Programme for Today

Epilog Editors

- Epilog inputs and outputs
- Tables, Charts, Graphs
- Specialized representations

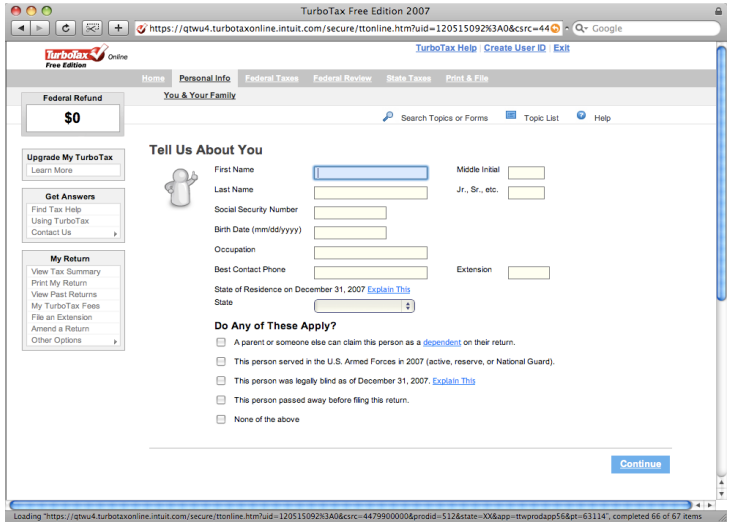
EpilogJS

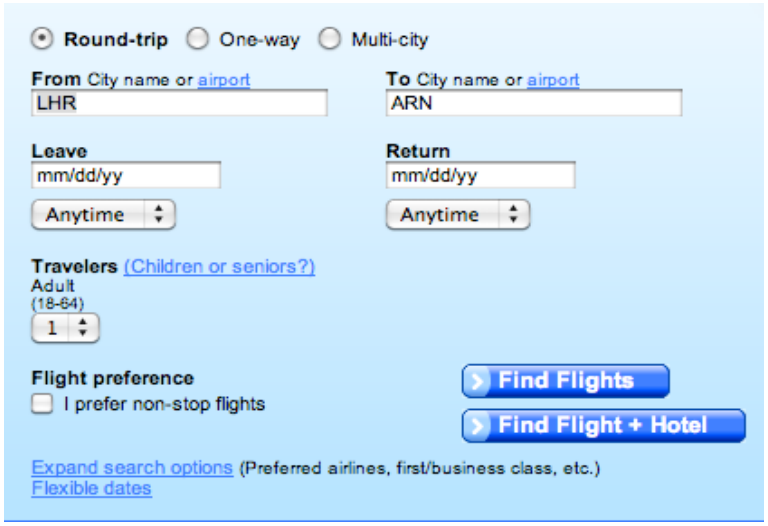
- Library of Javascript subroutines for managing Epilog
- Useful for building custom systems

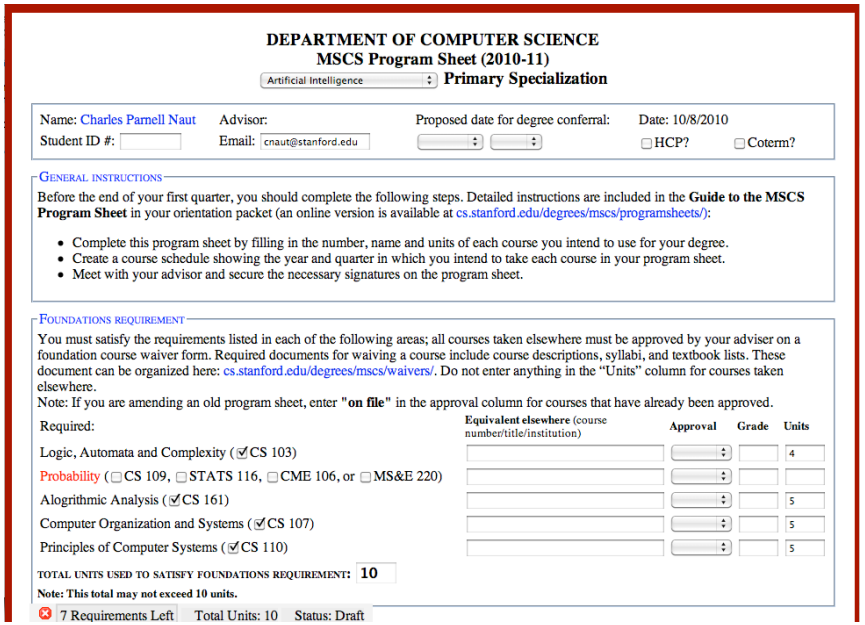
EpilogJS Interpreters

- Systems that run Javascript and EpilogJS

Next Time - Worksheets







Epilog Editors

Sierralite

Sierralite

What versus How

Rules | Data | Evaluate | Query | Execute | About

Query

Pattern goal(X,Y,Y,X)

Query p(X,Y)

Show Results 100 Unification Limit 1000000 Autorefresh

4 unification(s)

```
goal(a,b,b,a)
goal(b,c,c,b)
goal(c,d,d,c)
goal(d,e,e,d)
```

Data

Update | Revert | Load | Save

```
p(a,b)
p(b,c)
p(c,d)
p(d,e)
```

Load Configuration | Save Configuration

Demonstration

Relations as Tables

Sierratable

What
versus
How

Rules | Data | Evaluate | Query | Execute | About

Table

Pattern goal(X,Y,Y,X)
Query p(X,Y)
Show Results 100 Unification Limit 1000000 Autorefresh

4 unification(s)

a	b	b	a
b	c	c	b
c	d	d	c
d	e	e	d

Data

Update | Revert | Load | Save

```
p(a,b)
p(b,c)
p(c,d)
p(d,e)
```

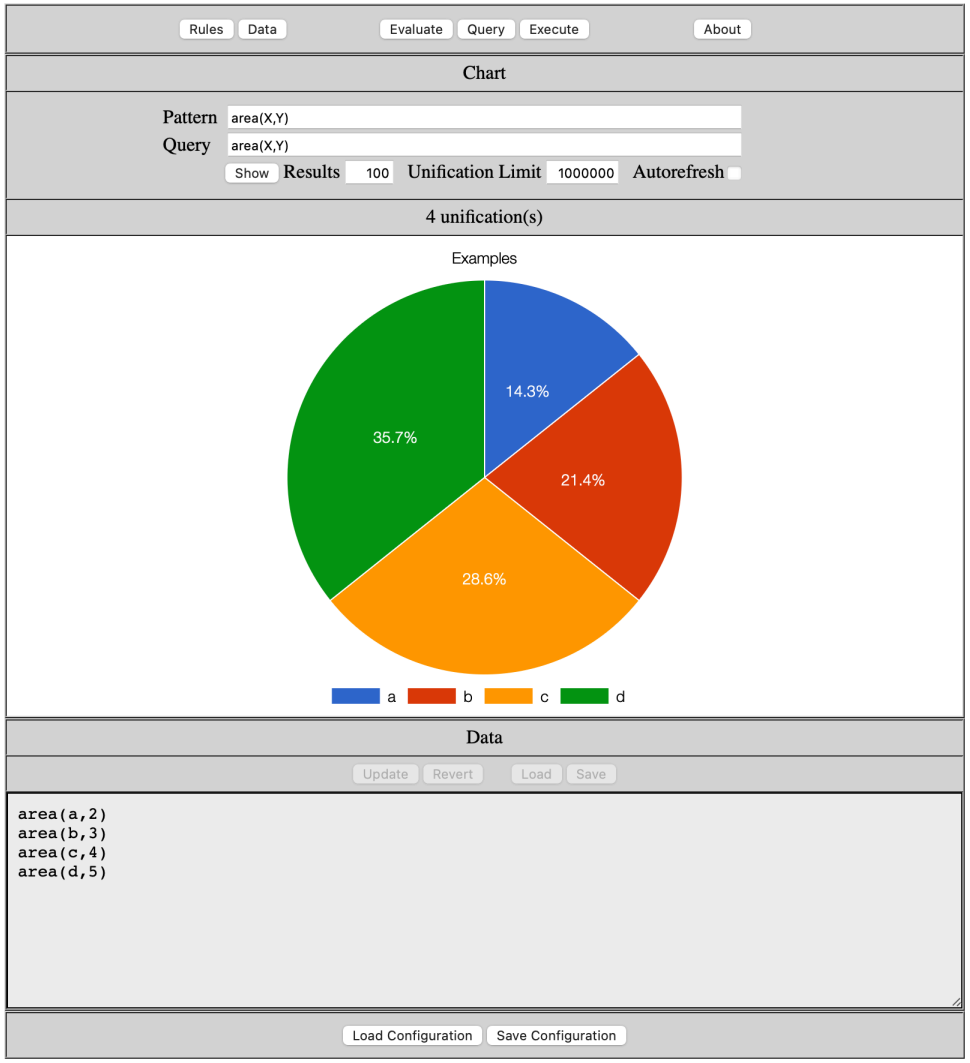
Load Configuration | Save Configuration

Demonstration

Numerical Functions as Pie Charts

Sierrachart

What versus How



Demonstration

Logic Graphs

Sierragrid

What versus How

Rules | Data | Evaluate | Query | Execute | About

Grid

Pattern p(X,Y)

Query p(X,Y)

Show Results 100 Unification Limit 1000000 Autorefresh ☑

6 unification(s)

	b	c	d
a	✓		✓
b		✓	
c	✓	✓	✓

Data

Update | Revert | Load | Save

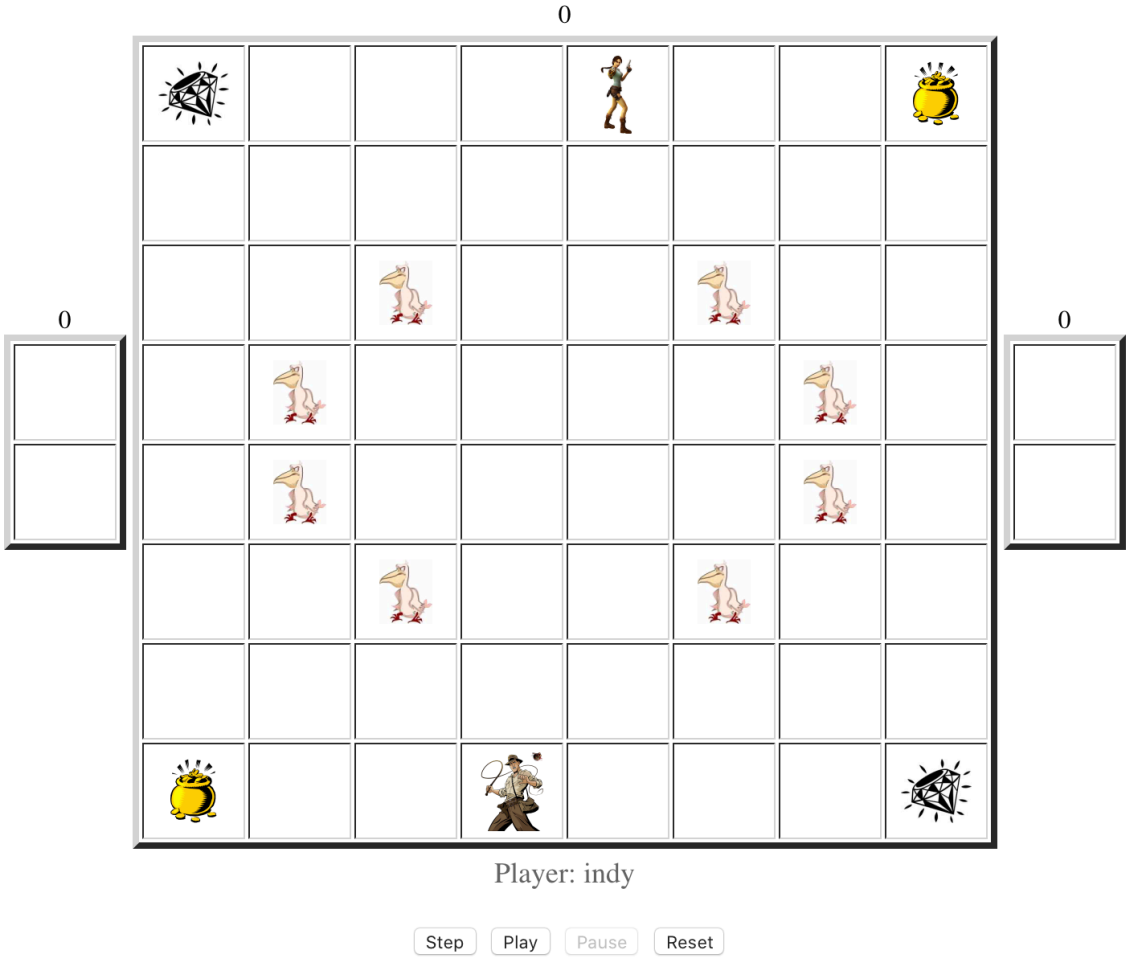
```
p(a,b)
p(b,c)
p(c,d)
p(c,b)
p(c,c)
p(a,d)
```

Load Configuration | Save Configuration

Demonstration

Pelican Hunters

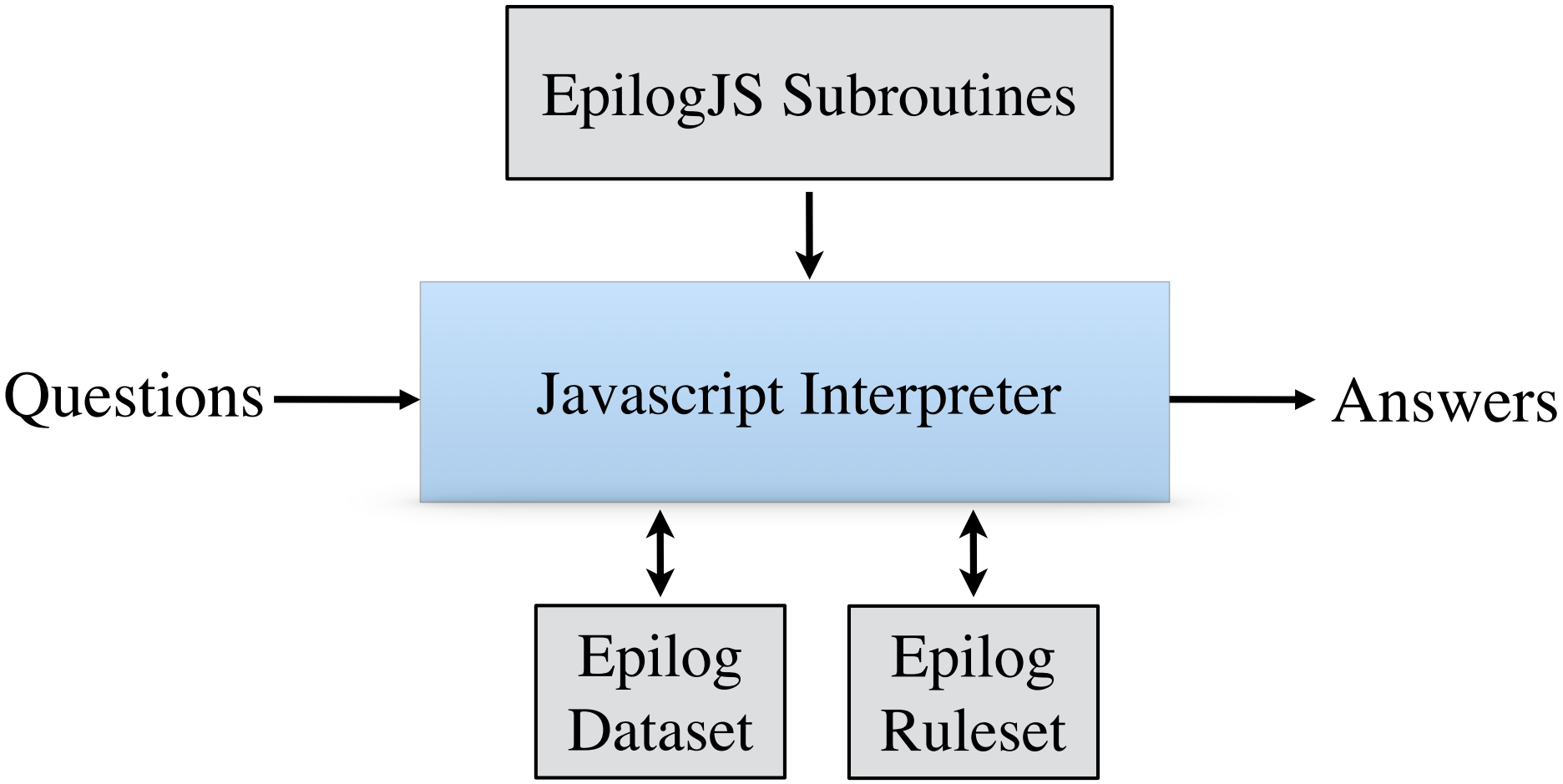
Pelican Hunters



Demonstration

EpilogJS

Javascript Implementation



Strings vs Expressions

Epilog expressions are written as strings of characters.
EpilogJS subroutines operate on parsed versions.

Examples:

```
p(a,b)
[p,a,b]

gp(X,Z) :- p(X,Y) & p(Y,Z)
[rule,[p,X,Z],[p,X,Y],p[Y,Z]]

invert :: p(X,Y)) ==> ~p(X,Y) & p(Y,X)
[handler,
 invert,
 [transition,[p,X,Y],
             [and,[not,[p,X,Y]],[p,Y,X]]]
```

Conversion Between Strings and Expressions

`read` transforms strings into expressions:

```
Call: read("gp(X,Z) :- p(X,Y) & p(Y,Z)")
Exit: [rule,[p,X,Z],[p,X,Y],p[Y,Z]]
```

`grind` transforms expressions into strings:

```
Call: grind([rule,[p,X,Z],[p,X,Y],p[Y,Z]])
Exit: gp(X,Z) :- p(X,Y) & p(Y,Z)
```

Datasets and Rulesets

A dataset is a linear array of data expressions.

```
[[p,a,b],[p,b,c],[p,c,d]]
```

A ruleset is a linear array of rules expressions.

```
[[rule,[g,X,Z],[p,X,Y],p[Y,Z]],
 [rule,[s,Y],[p,X,Y]]]
```

Conversion between Strings and Datasets

`readdata` transforms strings into datasets:

```
Call: readdata("p(a,b) p(b,c) p(c,d)")
Exit: [[p,a,b],[p,b,c],[p,c,d]]
```

`grindem` transforms datasets into strings:

```
Call: grindem([[p,a,b],[p,b,c],[p,c,d]])
Exit: p(a,b) p(b,c) p(c,d)
```

Same for rulesets

Data Indexes

In *data indexing*, each factoid appears on the list of factoids associated with each constant in that factoid.

Example: `{p(a,b), p(b,c), q(b), q(c)}`

Index on `p`: `{p(a,b), p(b,c)}`
Index on `q`: `{q(b), q(c)}`

Index on `a`: `{p(a,b)}`
Index on `b`: `{p(a,b), p(b,c), q(b)}`
Index on `c`: `{p(b,c), q(c)}`

NB: No *compound* indices (e.g. all factoids with `a` *and* `b`)
NB: No *positional* indices (e.g. `p(a,b)` vs. `p(b,a)`)

Data Indexing

`definefacts(`*dataset*,*facts*`)` empties *dataset* and adds *facts* together with full indices.

```
Call: definefacts(lambda,read("p(a,b) p(b,c)"))
Exit: true

Call: lambda
Exit: [[p,a,b],[p,b,c]]
```

and all facts are data indexed

Rules Indexes

In *rule indexing*, each rule appears on the list of rules associated with relation constant in the head of the rule.

Example: {`r(X):-q(X,Y), q(X,Y):-p(X,Y)`}

Index on `p`: {}
Index on `q`: {`q(X,Y):-p(X,Y)`}
Index on `r`: {`r(X):-q(X,Y)`}

Rule Indexing

`definerules(`*ruleset*`,`*rules*`)` empties *ruleset* and adds *rules* together with rule indices.

```
Call: definerules(library,readdata("r(X):-q(X,Y)"))
Exit: true

Call: library
Exit: [[p,a,b],[p,b,c]]
```

and all rules are rule indexed

Saving and Dropping Facts

`savefact(`*fact*`,`*dataset*`)` - adds *fact* to *dataset* and adds to the corresponding data index.

```
lambda = [[p,a,b],[p,b,c]]

Call: savefact(read("p(c,d)"),lambda)
Exit: [p,c,d]

Call: lambda
Exit: [[p,a,b],[p,b,c],[p,c,d]]
```

`dropfact(`*fact*`,`*dataset*`)` - drops *fact* from *dataset* and removes from the corresponding data index.

```
Call: dropfact(read("p(b,c)"),lambda)
Exit: [p,b,c]

Call: lambda
Exit: [[p,a,b],[p,c,d]]
```

Saving and Dropping Rules

`saverule(`*rule*`,`*ruleset*`)` - adds *rule* to *ruleset* and adds to the corresponding rule index.

```
library = {r(X):-q(X,Y)}
```

```
Call: saverule(read('q(X,Y):-p(X,Y)'),library)
```

```
Call: library
Exit: {r(X):-q(X,Y)} q(X,Y):-p(X,Y)}
```

`droprule(`*rule*`,`*ruleset*`)` - drops *rule* from *ruleset* and removes the corresponding rule index.

```
library = {r(X):-q(X,Y)} q(X,Y):-p(X,Y)}
```

```
Call: droprule(read('q(X,Y):-p(X,Y)'),library)
```

```
Call: library
Exit: {r(X):-q(X,Y)}
```

Lookup Subroutines

`compfindp(`*query*`,`*dataset*`,`*ruleset*`)` - returns `true` if the *query* is true given *dataset* and *ruleset*.

`compfindx(`*pattern*`,`*query*`,`*dataset*`,`*ruleset*`)` - returns an instance of *pattern* that makes the *query* true.

`compfinds(`*pattern*`,`*query*`,`*dataset*`,`*ruleset*`)` - returns all instances of *pattern* for which the *query* is true.

```
lambda = [[p,a,b],[p,b,c],[p,c,d]]
library = [[rule,[g,X,Z],[p,X,Y],p[Y,Z]]]

Call: compfindp('X',[g,X,Z]),lambda,library)
Exit: true

Call: compfindx('X',[g,X,Z],lambda,library)
Exit: a

Call: compfinds('X',[g,X,Z],lambda,library)
Exit: [a,b]
```

Update Subroutines

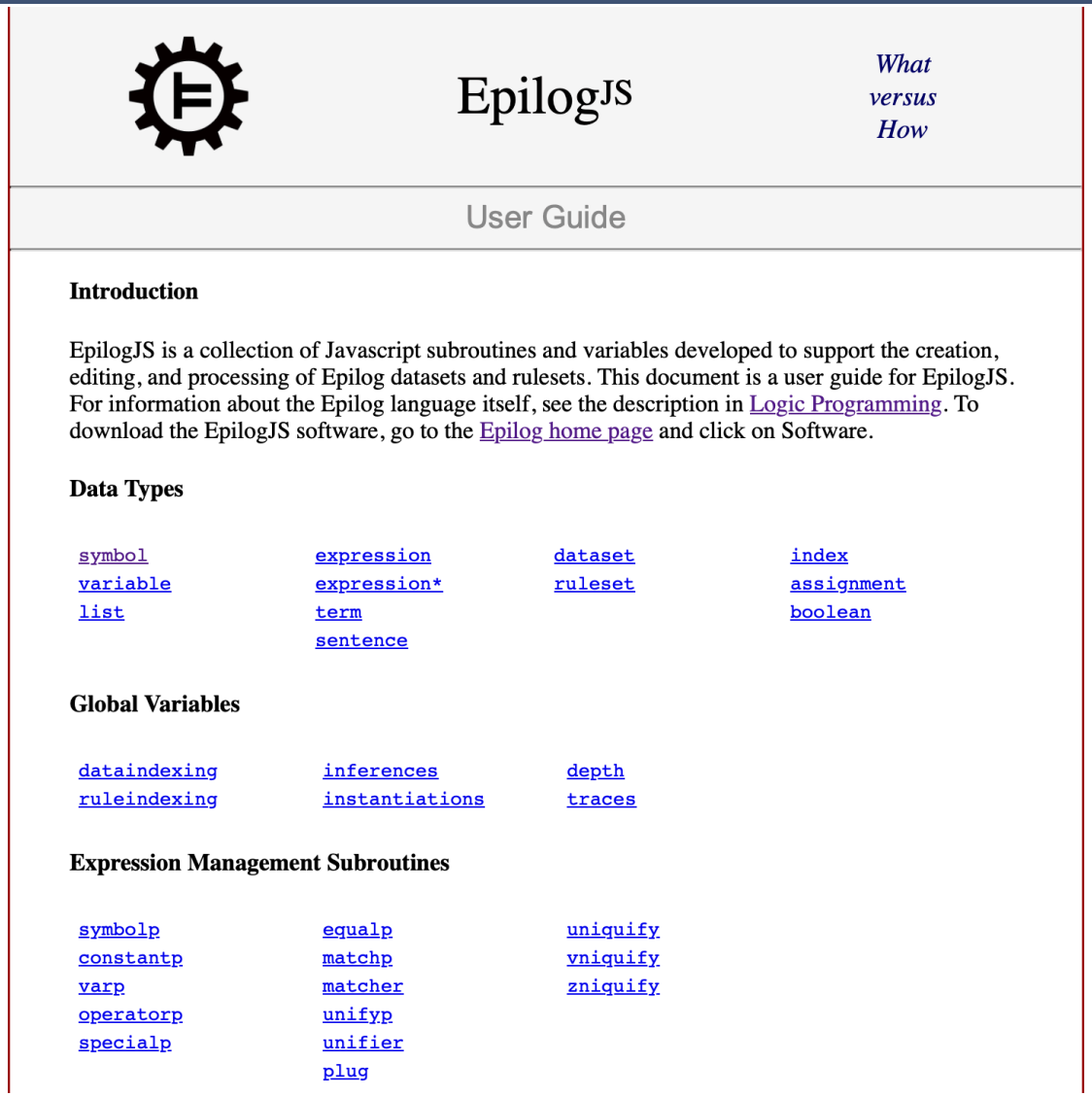
`compexpand(`*action*`,`*dataset*`,`*ruleset*`)` - returns the changes that would need to be made to execute *action* based *dataset* and *ruleset*.

```
lambda = [[p,a,b],[p,b,c]]
library = read("invert::p(X,Y) ==> ~p(X,Y)&p(Y,X)")

Call: compexpand('invert',lambda,library)
Exit: [[not,[p,a,b]],[not,[p,b,c]],[p,b,a],[p,c,b]]
```

`compexecute(`*action*`,`*dataset*`,`*ruleset*`)` - executes *action*, modifying *dataset* in accordance with definitions in *ruleset*.

```
Call: compexecute('invert',lambda,library)
Exit: true

Call: lambda
Exit: [[p,b,a],[p,c,b]]
```

Documentation



EpilogJS

What versus How

User Guide

Introduction

EpilogJS is a collection of Javascript subroutines and variables developed to support the creation, editing, and processing of Epilog datasets and rulesets. This document is a user guide for EpilogJS. For information about the Epilog language itself, see the description in Logic Programming. To download the EpilogJS software, go to the Epilog home page and click on Software.

Data Types

symbol	expression	dataset	index
variable	expression*	ruleset	assignment
list	term		boolean
	sentence		

Global Variables

dataindexing	inferences	depth
ruleindexing	instantiations	traces

Expression Management Subroutines

symbolp	equalp	uniquify
constantp	matchp	vniquify
varp	matcher	zniquify
operatorp	unifyp	
specialp	unifier	
	plug	

http://epilog.stanford.edu/documentation/epilogjs/guide.php

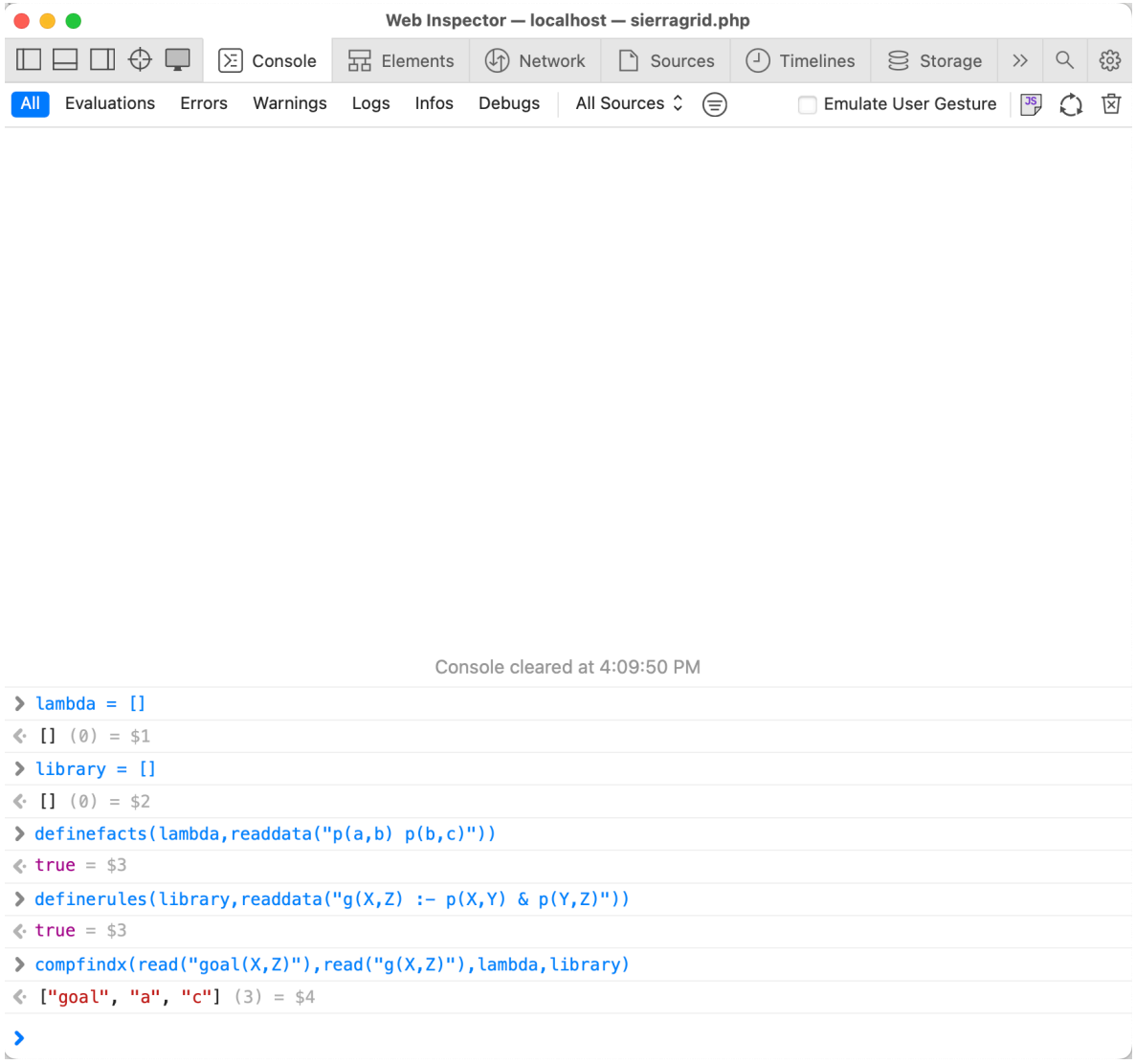
Source Code

```
//============================================================================
// Epilog
//============================================================================
// Copyright (c) 2020 The Board of Trustees of the Leland Stanford Junior
// University.  All nonprofit research institutions may use this Software for
// any non-profit purpose, including sponsored research and collaboration.  All
// nonprofit research institutions may publish any information included in the
// Software.  This Software may not be redistributed.  It may not be used for
// commercial purposes.  For any questions regarding commercial use or
// redistribution, please contact the Office of Technology Licensing at Stanford
// University (info@otlmail.stanford.edu).
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS";
// AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
// ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
// LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
// CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
// SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
// INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
// CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
// ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.
//============================================================================
// Sentential Representation
//============================================================================

function symbolp (x)
 {return typeof x==='string'}

function varp (x)
 {return (typeof(x)==='string' && x.length!==0 &&
          (x.charCodeAt(0)===95 || x[0]!==x[0].toLowerCase()))}

function constantp (x)
 {return typeof x==='string' && x.length!==0 && x[0]===x[0].toLowerCase()}

function numberp (x)
 {return !isNaN(Number(x))}

function stringp (x)
 {return typeof x==='string' && x.length>1 && x[0]==='"' &&  x[x.length-1]==='"'}

var counter = 0

function newvar ()
 {counter++;  return 'V' + counter}

function newsym ()
 {counter++;  return 'c' + counter}

function seq ()
 {var exp=new Array(arguments.length);
  for (var i=0; i<arguments.length; i++) {exp[i]=arguments[i]};
  return exp}
```

http://epilog.stanford.edu/javascript/epilog.js

EpilogJS Interpreter

EpilogJS Interpreter



Demonstration