Logic Programming
Functions

Michael Genesereth
Computer Science Department
Stanford University

Compound Names

Food World

```
three(calamari,beef,shortcake)
four(consomme,greek,lamb,baklava)
five(vichyssoise,caesar,trout,chicken,tiramisu)
```

Linked Lists

```
cons(a,cons(b,cons(c,cons(d,nil))))
```

Predefined Functions

Arithmetic

```
plus(2,3)  --> 5
times(2,3) --> 6
```

Lists

```
length([1,2,3]     --> 3
reverse([a,b,c])   --> [c,b,a]
append([a,b],[c]) --> [a,b,c]
```

Conditionals and Selections and Aggregates

```
if(p(a),1,true,2)  --> 1
choose(X,p(X))     --> a
countofall(X,p(X)) --> 3
setofall(X,p(X))   --> [a,b,c]
```

User-Defined Functions

Geometry

```
area(rectangle13) --> 6
```

Lists

```
rev([a,b,c,d]) --> [d,c,b,a]
```

Tree

```
height(cons(cons(a,b),cons(c,d))) --> 2
```

Matrices

```
transpose([[a,b],[c,d]]) --> [[a,c],[b,d]]
```

Function Definitions

View Definitions and Function Definitions

View Definitions

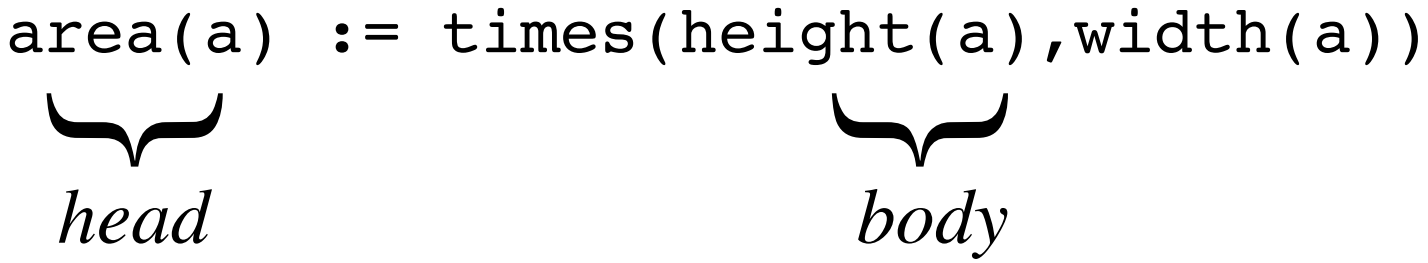
```
r(X,Y) :- p(X,Y) & ~q(Y)
s(X,Y) :- r(X,Y) & r(Y,Z)
```

Function Definitions

```
app(nil,Z) := Z
app(cons(X,Y),Z) := cons(X,app(Y,Z))

area(X) := times(height(X),width(X))
abs(X) := if(less(X,0),X,true,minus(0,X))
```

Function Definition



Variables

```
area(X) := times(height(X),width(X))
```

Safety

A function definition rule is *safe* if and only if every variable that appears on the right side appears in the head or is bound by a quantifier.

Safe:

```
h(X,Y) := g(f(X),f(Y))
h(X,Y) := if(p(X),X,Y)
h(X,Y) := choose(Z,p(Z))
h(X,Y) := countofall(Z,p(X,Z))
```

Unsafe:

```
h(X,Y) := g(f(X),Z)
```

Useful Exception

A function definition rule is *safe* if and only if every variable that appears on the right side appears in the head or is bound by a quantifier.

Not Safe but Okay:

```
h(X) := if(p(X,Y),Y,X)
```

Example:

```
{p(a,b),p(b,c)}
h(X) := if(p(X,Y),Y,X)

h(a) --> b
h(b) --> c
h(c) --> c
```

Sentences Inside Evaluable Terms

Dataset `{p(a,b),p(a,c),p(b,d)}`

Example

```
if(exist(Y,p(a,Y)&p(Y,c)),yes,no)
```

Result `yes`

Example

```
if(exist(Y,p(b,Y)&p(Y,c)),yes,no)
```

Result `no`

Evaluate Predicate

`evaluate(x,v)`

x is a term

v is the "value" of x

Examples

```
goal :- evaluate(times(2,3),6)

goal :- evaluate(plus(times(2,3),4),10)

area(X,A) :-
  h(X,H) & w(X,W) & evaluate(times(H,W),A)
```

Safety: all variables in *first* argument must be bound.

Arithmetic

Numeric Functions

Factorial

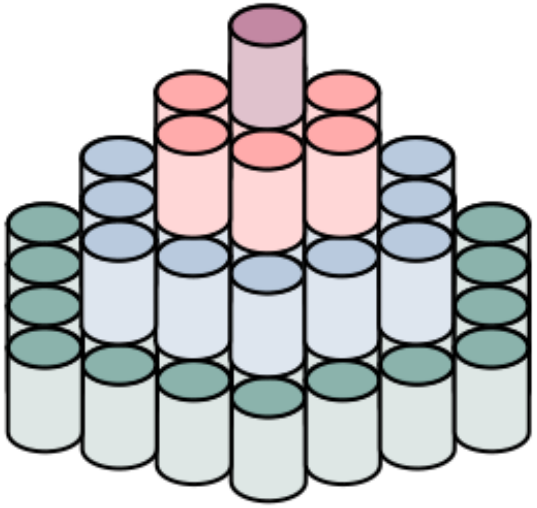
```
fact(0) := 1
fact(N) := times(N,fact(minus(N,1)))
```

Fibonacci

```
fib(0) := 1
fib(1) := 1
fib(N) := plus(fib(minus(N,1)),fib(minus(N,2)))
```

Cans

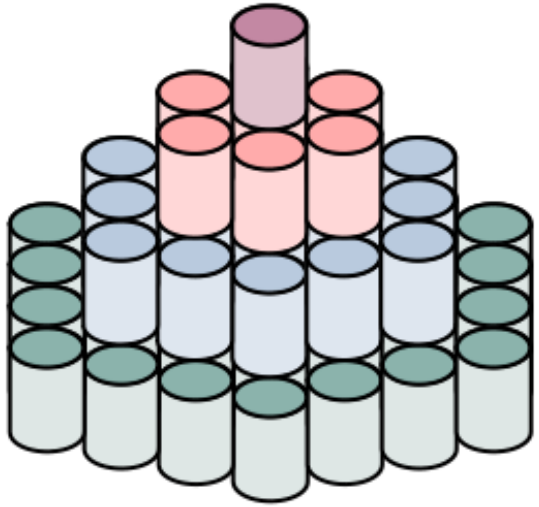
Cans



Solution

Number of cans per layer (starting at 1):

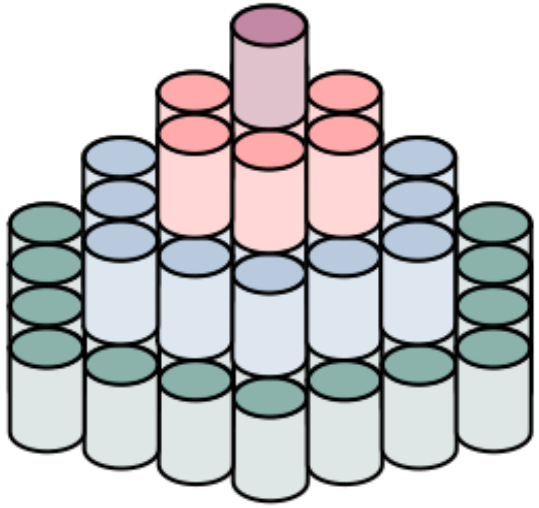
```
h(1) := 1
h(Z) := plus(h(minus(Z,1)),times(6,minus(Z,1)))
```



Solution

Number of cans per layer (starting at 1):

```
h(1) := 1
h(Z) := plus(h(minus(Z,1)),times(6,minus(Z,1)))
```



Non-recursive definition in terms of + and x:

```
h(Z) := plus(times(3,minus(Z,1),Z),1)
```

Lists, Sets, Trees

List Functions

Concatenation

```
  appfun(nil,Z) := Z
  appfun(cons(X,Y),Z) := cons(X,appfun(Y,Z))

  appfun(nil,Z) := Z
  appfun(X!Y,Z) := X!appfun(Y,Z)

appfun([a,b],[c,d]) --> [a,b,c,d]
```

Set Functions

Union

```
  union(nil,Z) := Z
  union(cons(X,Y),Z) :=
    if(member(X,Z),Z,cons(X,union(Y,Z)))

union([a,b],[c,d]) --> [a,b,c,d]
```

Intersection

```
  inter(nil,Z) := nil
  inter(cons(X,Y),Z) :=
    if(member(X,Z),cons(X,inter(Y,Z)),
       inter(Y,Z))

inter([a,b],[b,c]) --> [b]
```

Tree Functions

```
height(Z) :=
  if(evaluate(symbolize(stringify(Z)),Z),0)

height(cons(X,Y)) :=
  plus(max(height(X),height(Y)),1)


height(cons(cons(a,b),cons(c,d))) --> 2
```

Computation Cost

Relational versus Functional

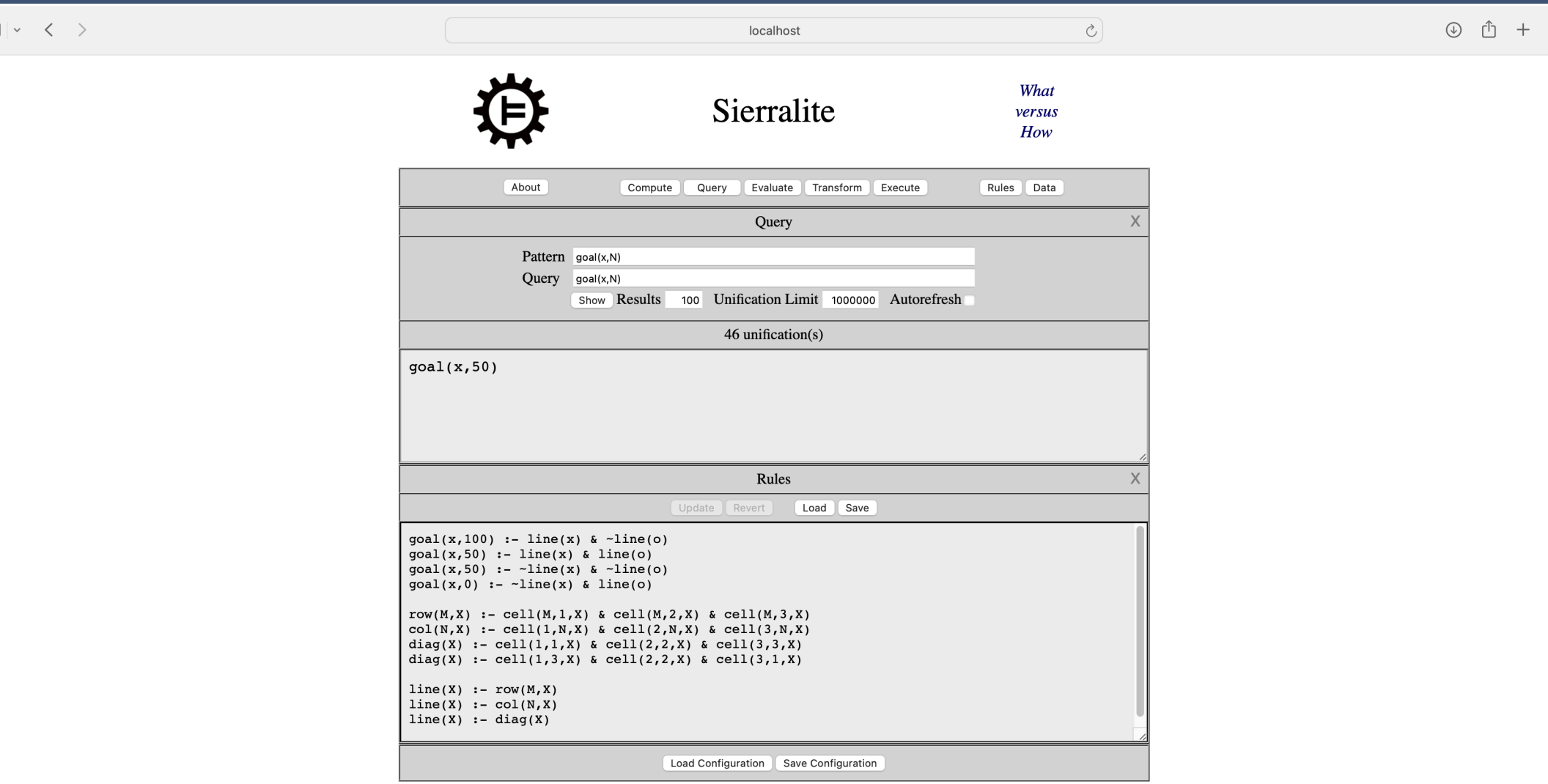
Relational Definition

```
goal(x,100) :- line(x) & ~line(o)
goal(x,50) :- line(x) & line(o)
goal(x,50) :- ~line(x) & ~line(o)
goal(x,0) :- ~line(x) & line(o)
```

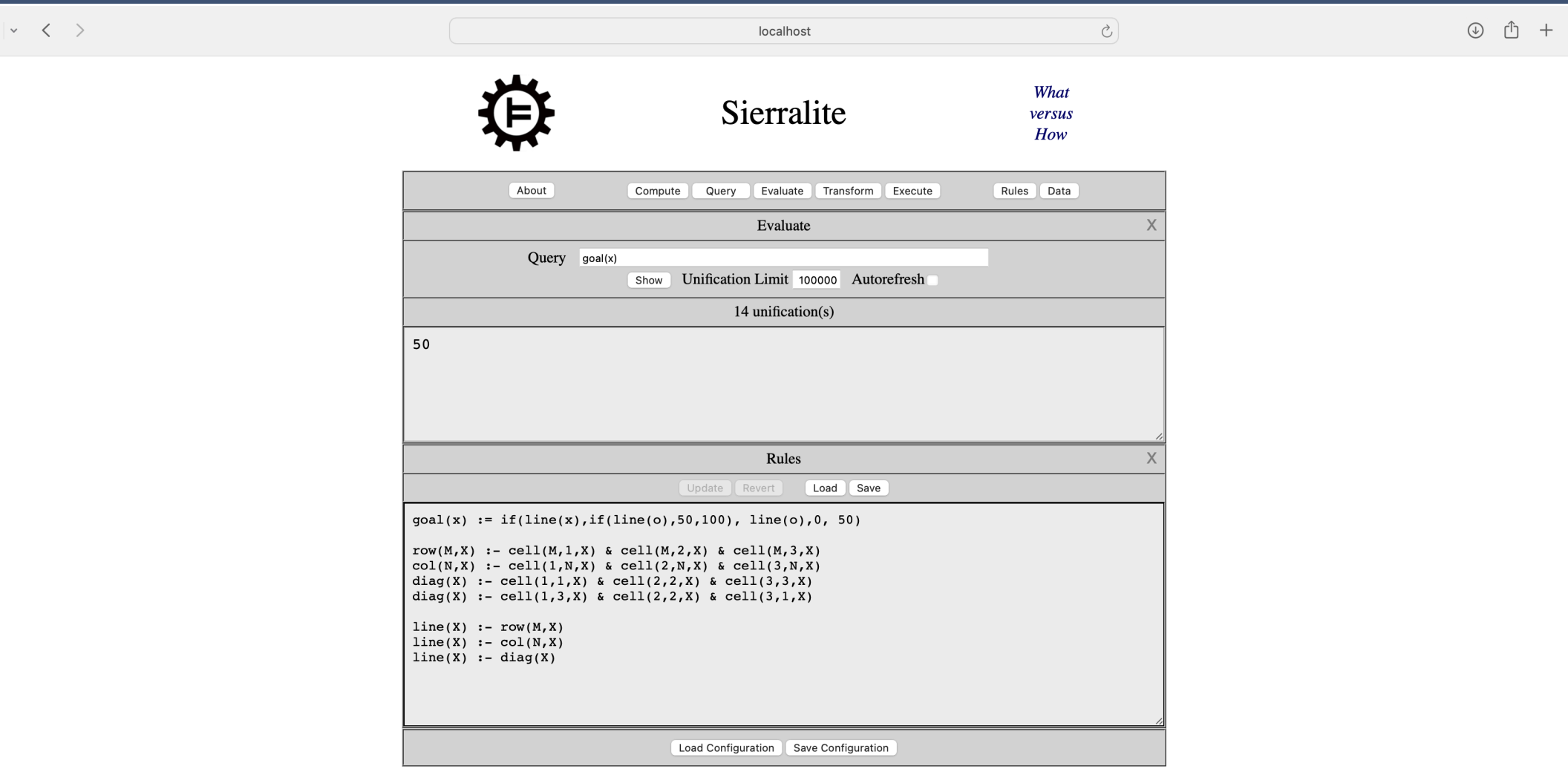
Functional Definition

```
goal(x) :=
  if(line(x),if(line(o),50,100),
     line(o),0,
     50)
```

Relational Cost



Functional Cost



Irreversibility

View Definitions and Function Definitions

View Definitions

```
apprel(nil,Z,Z)
apprel(cons(X,Y),Z,cons(X,W)) :-
  apprel(Y,Z,W)
```

Function Definitions

```
appfun(nil,Z) := Z
appfun(cons(X,Y),Z) := cons(X,app(Y,Z))
```

Reversibility

Relation Definition

```
apprel(nil,Z,Z)
apprel(cons(X,Y),Z,cons(X,W)) :-
  apprel(Y,Z,W)
```

Sample Queries and Results

```
apprel([a,b],[c,d],Z)  --> [a,b,c,d]
apprel(X,[c,d],[a,b,c,d])  --> [a,b]
apprel([a,b],Y,[a,b,c,d])  --> [c,d]
apprel(X,Y,[a,b,c,d])  -->
  [],[a,b,c,d]
  [a],[b,c,d]
  [a,b],[c,d]
  [a,b,c],[d]
  [a,b,c,d],[]
```

Irreversibility

Function Definition

```
appfun(nil,Z) := Z
appfun(cons(X,Y),Z) := cons(X,app(Y,Z))
```

Sample Queries and Results

```
appfun([a,b],[c,d])  --> [a,b,c,d]
apprel(X,[c,d])      --> false
apprel([a,b],Y)      --> false
apprel(X,Y,[a,b,c,d])--> false
```

No unbound variables in evaluable terms!